CASL-U-2015-0079-000

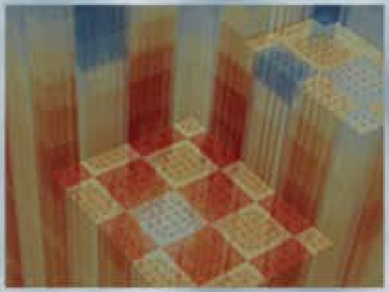
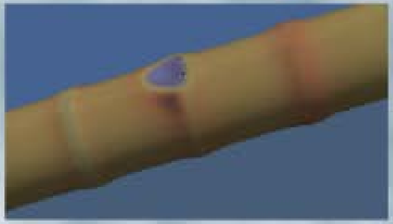
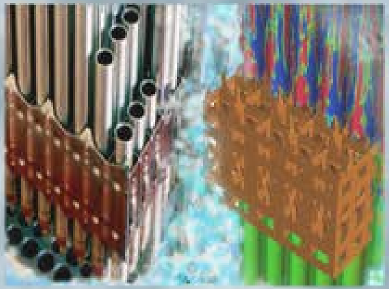
Exnihilo User's Manuals

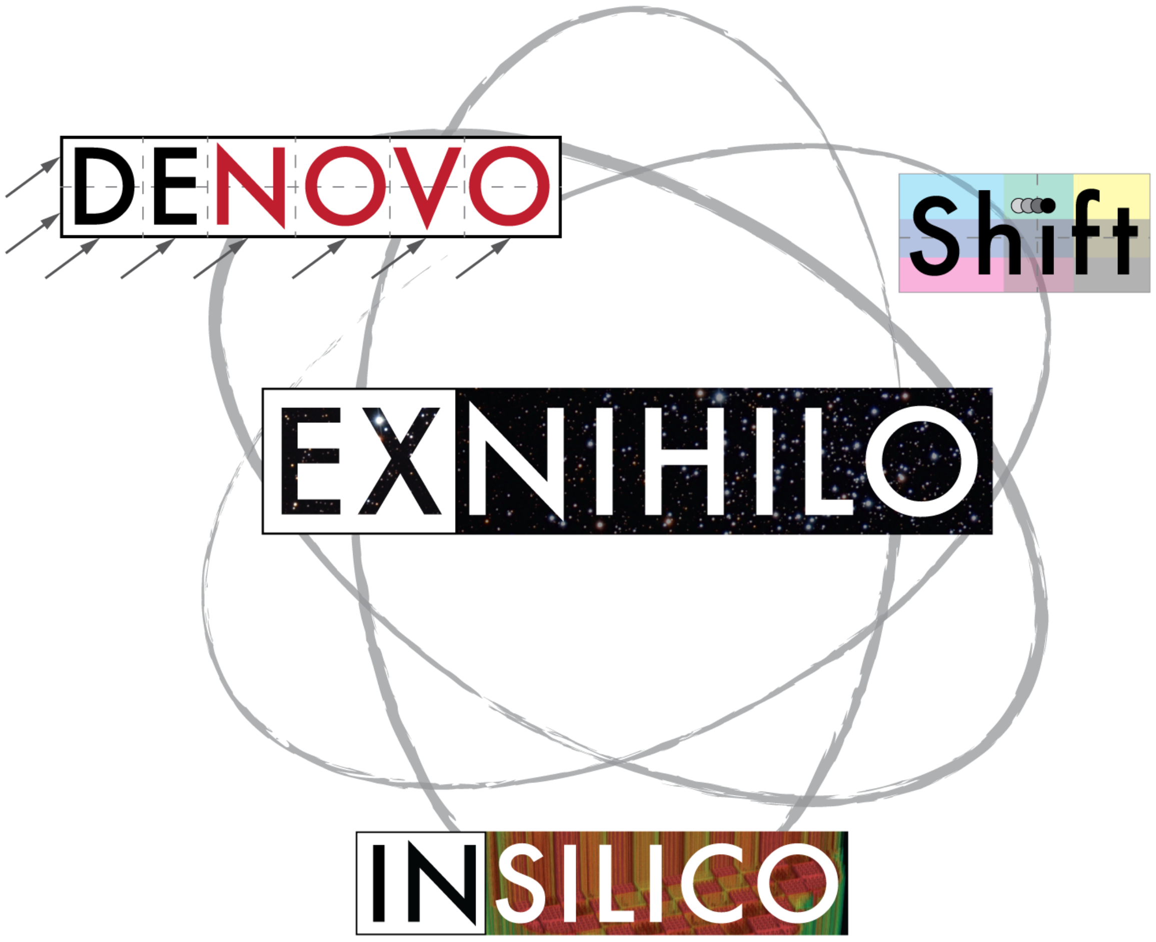
Release 5.3 (Dev)

Thomas Evans
Greg Davidson
Steven Hamilton
Seth Johnson
Tara Pandya

Oak Ridge National Laboratory

February 19, 2015



Exnihilo Documentation

Release 5.3 (Dev)

Seth Johnson Tom Evans Greg Davidson
Steven Hamilton Tara Pandya

February 19, 2015

CASL-U-2015-0079-000

Part I

Overview

CHAPTER

ONE

INTRODUCTION

Exnihilo is a modern radiation transport framework that implements a variety of advanced solvers and solution methodologies, enabling it to solve a wide variety of nuclear engineering and applications problems with the scalability to run on both desktop machines and leadership-class supercomputers.

It supports “stand-alone” execution using its internal front ends, but its components (such as the the Denovo S_N solver) are also integrated into other radiation transport codes.

1.1 Package structure

Exnihilo currently contains the following source code packages:

Nemesis: Infrastructure components This collection of utilities should be applicable to most scientific codes. Included are Design-by-Contract ™ utilities, communication libraries, containers and algorithms, HDF5 and Silo interface wrappers, template-based serialization utilities, and the unit testing harness.

Transcore: Tranport core components These components are specific to radiation transport and multiphysics codes. Included are cross section storage classes, libraries for reading and writing cross sections, quadrature sets, Trilinos solver wrappers, Monte Carlo samplers, and Python interface wrappers.

Geometria: Geometry packages The geometries in this class are used for meshing deterministic problems, for transporting Monte Carlo particles on, and for tallying. Geometries include mesh and cylindrical mesh, Reactor ToolKit geometry, MCNP geometry using the Lava wrapper, SCALE geometry, and DagMC geometry.

Physica: Physics packages These are geometry-agnostic physics engines used for Monte Carlo transport. Currently included are continuous-energy and multigroup physics packages.

Denovo: Deterministic transport solvers Denovo contains advanced transport solvers for fixed-source and eigenvalue problems, with discrete ordinates (S_N) and simplified spherical harmonics (SP_N), using Cartesian grids with the KBA decomposition.

Shift: Monte Carlo solver The Shift Monte Carlo framework is based on geometry- and physics-agnostic transport routines. These routines include advanced parallel algorithms, flexible tallies, and extensible source definitions.

Insilico: Neutronics front end The Insilico front end couples Denovo and Shift with cross-section processing, depletion, and thermo-hydraulics feedback for reactor analysis. This component is integrated into VERA, the Virtual Environment of Reactor Applications.

Omnibus: General front end Omnibus is an ASCII-driven front end to Exnihilo for general applications.

1.2 Development team

The core Exnihilo development team consists of the following ORNL scientists (listed alphabetically):

- Greg Davidson <davidsongg@ornl.gov>
- Tom Evans <evanstm@ornl.gov>
- Steven Hamilton <hamiltonsr@ornl.gov>
- Seth Johnson <johnsonsr@ornl.gov>
- Tara Pandya <pandyatm@ornl.gov>

Additional developers that have contributed to the code base to varying degrees and are active *associate developers* of Exnihilo are:

- Cihangir Celik <celikc@ornl.gov>
- Kevin Clarno <clarnokt@ornl.gov>
- Wayne Joubert <joubert@ornl.gov>
- Chris Perfetti <perfetticm@ornl.gov>
- Rachel Slaybaugh <slaybaugh@berkeley.edu>

Emeritus developers, who have moved on to other projects, include:

- Joshua Jarrell
- Brenden Mervin
- Stuart Slattery

General questions about Exnihilo software and methods should be directed to the email list.

CHAPTER

TWO

EXNIHILO QUICK START QUIDE

2.1 Building the developer documentation

Documentation in Exnihilo comes in three forms:

METHODS Papers, reports, and notes that describe numerical methods, algorithms, and results. These documents are prepared using the latex document system.

CODE MANUALS Manuals for code developers, the development environment, and users. This document falls into this category. We use the Python Sphinx system to produce multiple document formats (PDF, HTML, text, etc) from basic reStructured (rst) formatted markup.

CODE DOCUMENTATION Inline code documentation. This class of documentation is almost exclusively reserved for code developers. It is processed using the Doxygen inline code documentation system from comments that live within the source code.

Details on building **METHODS** and **CODE DOCUMENTATION** are found in *Developer Guide*. To build the full developer documentation Python must be available with the Python Sphinx module installed. Enter `Exnihilo/doc/manual` and run **make**:

```
$ make
Please use `make <target>' where <target> is one of
  html       to make standalone HTML files
  dirhtml    to make HTML files named index.html in directories
  singlehtml to make a single large HTML file
  pickle     to make pickle files
  json       to make JSON files
  htmlhelp   to make HTML files and a HTML help project
  qthelp     to make HTML files and a qthelp project
  devhelp    to make HTML files and a Devhelp project
  epub       to make an epub
  latex      to make LaTeX files, you can set PAPER=a4 or PAPER=letter
  latexpdf   to make LaTeX files and run them through pdflatex
  text       to make text files
  man        to make manual pages
  texinfo    to make Texinfo files
  info       to make Texinfo files and run them through makeinfo
  gettext    to make PO message catalogs
  changes    to make an overview of all changed/added/deprecated items
  linkcheck  to check all external links for integrity
  doctest    to run all doctests embedded in the documentation (if enabled)
```

Choose the desired format and run `make <format>`. The build result will be in `_build/`*`target`*.

2.2 Assembling the source code

Exnihilo integrates with the build system of three external code repositories: copies of Exnihilo, Trilinos, and TriBITS must be located inside of the SCALE source directory in order to build:

```
$ cd /usr/local/src
$ ls SCALE
CMakeLists.txt ... Exnihilo TriBITS Trilinos ...
```

See *Getting the code* for details.

2.3 Building the code

After the necessary *Third party libraries* have been built and installed, Exnihilo can be built with CMake. The most straightforward way to get started building Exnihilo is to use the scripts in `Exnihilo/install` that are described in *Configuration Management*. The `install.sh` script described in *The install.sh script* is used to manage the build process.

To build the base release of Exnihilo run the following:

```
./install.sh Exnihilo
```

To build only the Exnihilo components needed for ADVANTG, run:

```
./install.sh Exnihilo for-advantg
```

CHAPTER

THREE

SOFTWARE TESTING AND VERIFICATION

As described in *Developer Guide*, the software in Exnihilo is highly tested. Specifically, every class/component in the code is **required** to have an individually compiled unit test verifying software correctness. This strategy has the side benefit of creating simplified *use cases* for each class that can be used by novice code developers to learn the interface and function of each class in the code.

Naturally, unit testing is only one aspect of a complete testing framework that demonstrates code verification and validation (V&V). First, we define some terms that will establish the frame-of-reference for code activities.

Verification Given a well posed and defined problem, whether the code produces the correct solution within the constraints imposed by the algorithm.

Validation Whether the model problem solved by the software matches experimental data or other accepted standards.

In essence, **Verification** determines that a given problem is being solved correctly; **Validation** determines that the correct problem is being solved. Within the framework of these definitions, most testing in Exnihilo is focused on **Verification**, not **Validation**. We can ensure that the code produces the correct output for a given set of inputs; however, we cannot ensure that the methods are being applied appropriately to a given problem. That information must, by definition, be performed for a given application.

For example, the **Denovo** package contains a simplified harmonics solver (SP_N). This method can have issues in void and near-void regions. Thus, a user wishing to solve transport problems in an application space that includes voids should probably consider using another method. However, the fact that SP_N is inappropriate, or *unvalidated*, for this class of problems does not negate the fact that the **Denovo** SP_N solver is **Verified**, ie. the weakness of SP_N for this class of problems lies with the method and model, not in the manner in which it is solved. In other words, the fitness of a given model, in this case SP_N, to a given problem is determined by **Validation**.

In addition to unit tests, Exnihilo contains multiple *acceptance* tests that compare against reference solutions, either analytical or from other validated codes. These are described in *Shift Acceptance Test Descriptions* and *Denovo Acceptance Test Descriptions*.

Part II

Developer Guide

CHAPTER

FOUR

INTRODUCTION

The developer's guide lays out how to obtain Exnihilo, how to install it, and how to successfully develop for it.

1. Read *Getting the code* to learn how to access the repositories that contain Exnihilo and associated software.
2. Read the *Exnihilo Installation Guide* guide, installing any TPLs as necessary and then installing Exnihilo itself.
3. Install the Exnihilo *Development Environment* to set up the environment template and scripts.
4. Watch the Lego Movie and possibly some Sesame Street while everything installs.
5. Get to work!

CHAPTER

FIVE

CONFIGURATION MANAGEMENT

This section describes obtaining and configuring:

Exnihilo and build system Exnihilo must be built alongside SCALE reactor analysis criticality tools, Trilinos numerical software, and TriBITS build tools.

TPLs The third-party toolchain and libraries listed in *Third party libraries* that are used to build, run, and test the code.

Toolchain Software utilities that are used to build, edit, profile, document, and perform configuration management activities.

We provide scripts for building and setting up the TPLs and Toolchain in `scripts`. See *Configuration Management* for details on using these scripts. Instructions for additional *Useful tools* is available later.

5.1 Getting the code

Exnihilo uses the TriBITS build system, it also has a required dependency on Trilinos and SCALE, requiring all four code systems to be downloaded and built at once. Note that while the existence of these systems is required, only the requested parts of each code will be built.

Note: If building from the ADVANTG or SCALE distributions, all of these source packages are already available and in the correct locations. You may skip to the next section.

5.1.1 Exnihilo

Configuration management in Exnihilo is managed using **git**. The source repository is stored on the ORNL machine angband.ornl.gov:

```
git clone ssh://angband.ornl.gov/repos/git/Exnihilo.git Exnihilo
```

You will need to contact the Exnihilo Team to get access to the repository.

5.1.2 SCALE

It is also necessary to download SCALE to build Exnihilo. SCALE is managed using Mercurial and is cloned using:

```
hg clone https://fogbugz.ornl.gov/kiln/RepoAlias/scale/scale Scale
```

For access to the RNSD fogbugz server, contact Jordan Lefebvre. To clone from fogbugz, you might also need to add the following to your `~/.hgrc`

```
[hostfingerprints]
fogbugz.ornl.gov = 39:81:1f:f4:82:5e:2a:64:7c:e2:6e:33:16:30:7d:be:43:e4:d3:3f
```

Note: If building for ADVANTG or Denovo solver development, it is possible to use a skeleton SCALE build system in lieu of the SCALE download above. This small "SCALE" directory is distributed with ADVANTG and only contains the essential CMake components of SCALE. (See *make-skeleton-scale* for details of the tool used to build it.)

SCALE data (necessary for running Insilico, Shift with CE, and other parts of the code) can be copied from the RNSD servers over SCP. We provide an *update-scale-data* tool to simplify loading and updating the data.

5.1.3 Trilinos and TriBITS

Finally, Trilinos and the TriBITS build system can be cloned from the angband server if at ORNL:

```
git clone ssh://angband.ornl.gov/repos/mirror/Trilinos.git Trilinos
git clone ssh://angband.ornl.gov/repos/mirror/TriBITS.git TriBITS
```

If access to the ORNL server is unavailable, Trilinos and TriBITS can be downloaded or cloned from their respective web sites.

5.1.4 Assembling the repositories

Exnihilo installation is set up with SCALE as the primary "source" directory; the Exnihilo, TriBITS, and Trilinos directories must be moved or symbolically linked inside it:

```
cd Scale
ln -s ../TriBITS
ln -s ../Trilinos
ln -s ../Exnihilo
```

5.2 Compilers and platforms

Currently, Exnihilo is tested on Linux and Mac OS X systems using both Intel and GCC compilers. Intel 14.0 and GCC 4.6 are the oldest supported versions due to the C++11 support requirement. A limited build of Exnihilo has been tested on Windows, but Windows is currently only supported as far as is required to run SCALE. Exnihilo is also compatible with Clang (i.e., the LLVM compiler on Apple systems), using `gfortran` to compile the Fortran code in Trilinos, Exnihilo, and SCALE.

5.3 Installation toolchain

To configure, install, and run all the components of Exnihilo, several tools need to be installed.

Table 5.1: Exnihilo toolchain

Tool	Required	Comments
CMake	Yes	Build system used by TriBITS to generate Makefiles.
Python	Not exactly	Needed for some advanced software configuration, for Omnibus, and for Python front end and tools.
SWIG	No	Needed for Python front end, with PCRE as a prerequisite TPL.

5.4 Third party libraries

Exnihilo makes use of several Third Party Libraries (TPLs). TPLs are distinct from *toolchain* components that are used to build, debug, profile, and test the code (e.g. the GCC compiler and Python). Libraries provide features to the compiled code. Examples include BLAS/LAPACK and HDF5. Some TPLs are provide both libraries and toolchain components, e.g. MPI. The Exnihilo toolchain is described in *Development Environment*. The following table gives a listing of the TPLS used by Exnihilo.

Table 5.2: Exnihilo TPLs

TPL	Required	Comments
MPI	No	OpenMPI and MPICH are suggested options. Vendor-provided options will also work.
QT	No	This is required by Scale. Any Exnihilo dependencies that require Scale will require QT.
BLAS/LAPACK	Yes	Use vendor-specific implementations when possible. The default implementations can be found at netlib.
HDF5	No	Both serial and parallel HDF5 implementations are supported. When building serial HDF5, parallel I/O using HDF5 is not available.
Silo	No	HDF5 is required when using Silo.
Lava	No	ORNL-developed interface library to MCNP services. Requires access to the MCNP source. Contact the Exnihilo team for access.

In addition to the above TPLs, Exnihilo supports many TPLs that are supported by Trilinos (e.g. SuperLU) solvers. See the `Trilinos build manual` for details.

Generic directions for building the TPLs are provided in *Development Environment*. Specific instructions for systems that differ from the general instructions are provided in *Exnihilo Installation Guide*.

5.4.1 Python TPLs

The following Python package are also recommended:

Table 5.3: Exnihilo Python packages

TPL	Required	Comments
Numpy	No	Advanced Python numeric manupulation, used by Python tools and Omnibus postprocessing.
h5py	No	HDF5 reading and writing in Python.
pandas	No	Python Data Analysis Library, used in some Omnibus postprocessing.

CHAPTER

SIX

EXNIHILO INSTALLATION GUIDE

This chapter documents a complete, from-scratch installation of Exnihilo. Most users will not need to read or understand this chapter, and even most developers (assuming they're on a pre-configured server) will not most of this reference. We provide detailed steps for most of the typical use cases.

6.1 Installation environment setup

Included in the `Exnihilo/install` directory is a set of build scripts developed to build Exnihilo, ADVANTG, and related prerequisites on various Linux and Mac systems. As the number of permutations of systems and codes and options has increased, the compilation scripts were restructured to allow for more flexibility. The file listing is:

`codes` cmake configurations and install scripts for relevant codes

`rc` environemnt settings for each platform/system (see *Install environment description*)

`tools` collection of scripts used for building and running regression tests

`install.sh` generic installer driver (see description and options below)

`ctest-driver.sh` testing/install driver for SCALE/Exnihilo

`README.rst` this readme file

`setup_macports.sh` recommended initial MacPorts installation script (see *Build toolchain with MacPorts*)

The recommended install process for a new system starts by setting up the installation environment for your machine:

```
cd Exnihilo/install/rc
cp -R $(uname) $(hostname -s)
cd $(hostname -s)
$EDITOR base.sh
```

Note: On my mac (Darwin system) with hostname `sierrajuliet`, the above simply evaluates to:

```
cp -R Darwin sierrajuliet
cd sierrajuliet
mvim base.sh
```

The scripts inside these files should be modified to reflect the base path for your source files, the path to SCALE, MCNP, and/or ADVANTG data files, etc. For a description of their contents, see *Install environment description*.

6.2 Install

Depending on what system you're using, and how customized you need your build, you have several options of what TPLs and tools need to be installed. We give several typical use cases below.

6.2.1 Building with Macports GCC

This section applies to most of the Exnihilo users at ORNL, but it could also be adapted for users with advanced package management systems.

1. *Build toolchain with MacPorts* to install GCC and the most of the TPLs.
2. Install Lava, the only TPL you must compile yourself, if using ADVANTG or Exnihilo with MCNP geometry support:

   ```
   cd Exnihilo/install
   ./install.sh lava
   ```

   You can also install other TPLs such as `MOAB` for CAD geometry support:

   ```
   ./install.sh moab
   ```

3. *Build Exnihilo from source*
4. If using ADVANTG, you can now install it too:

   ```
   ./install.sh advantg
   ```

6.2.2 Building Exnihilo on Titan

The Titan system uses the `module` system to load package support. Several packages must be loaded (preferably in your `.bashrc`) for TPLs and the GCC build chain to work:

```
module load gcc
module load git
module load mercurial
module load cmake3
module load python
module load cray-hdf5
module load cudatoolkit
module load python_numpy
module load python_h5py
module load python_matplotlib
```

The Titan system configure directory has variants for each project allocation. These can be used like:

```
./install -t nfi004 Exnihilo scale-debug
```

Note that to run unit tests and executables on Titan, it's necessary to log in to a compute node and execute the process using the `aprun` command.

6.2.3 Building the entire toolchain from scratch

If you're on a brand-new system using outdated software (such as Red Hat, whose compilers are frozen at several years behind the cutting edge for the sake of stability), it's typical to install the entire toolchain manually.

1. *Build GCC from source* to install the compilers.
2. *Build tools from source* for other configuration requirements.
3. *Build BLAS/LAPACK from source* if needed.
4. *Build public TPLs from source* for other Exnihilo requirements.
5. Install Lava and other feature-specific TPLs:

   ```
   ./install.sh lava
   ```

6. *Build Exnihilo from source*
7. If using ADVANTG, you can now install it too:

   ```
   ./install.sh advantg
   ```

6.3 Build steps

To install Exnihilo, it is often best to start building the compilers and third party libraries (TPLs) from scratch: the Python front-end `pykba`, because it relies on shared libraries, requires that all the prerequisite libraries be compiled with position-independent code (`-fPIC` option for GCC). This is most easily accomplished by building libraries as shared to begin with.

Installation works best with a recent build of the GNU compiler collection (gcc). On a Mac system, this is most easily accomplished using the MacPorts package distribution system. On old Linux systems (e.g. RHEL 4), it is best to download the prerequisite GCC libraries, build them, build and install GCC, and then install the remaining packages. Newer Linux systems (e.g. Ubuntu) often have up-to-date installations of GCC as well as package managers that can ease the installation process.

6.3.1 Build GCC from source

If building GCC from scratch on a Linux box, download these libraries into your source base directory (e.g. /usr/local/src):

- mpfr : GNU MPFR Library
- gmp : The GNU Multiple Precision Arithmetic Library
- mpc : The multiprecision library
- gcc : GNU compiler collection

To install GCC using the `bootstrap.sh` configuration you created:

```
./install.sh -t bootstrap gmp
./install.sh -t bootstrap mpfr
./install.sh -t bootstrap mpc
./install.sh -t bootstrap gcc
```

This builds a current version of GCC using whatever older version is available on the system.

6.3.2 Build tools from source

If a good package manager is unavailable on your system, you'll have to download and install the CMake, SWIG, and Python tools.

- cmake : CMake build system
- pcre : PCRE - Perl Compatible Regular Expressions
- swig : Simplified Wrapper and Interface Generator (SWIG)
- python : Python 2

After downloading, you can install these simply with:

```
./install.sh cmake
./install.sh pcre
./install.sh swig
./install.sh python
```

6.3.3 Build BLAS/LAPACK from source

Most systems include a compiled linear algebra library. (On Titan, these are embedded in the Cray scientific libraries; on a Mac and most Linux systems, they're located in /usr/lib.) If your machine does not, or you wish to have an autotuned library for your specific machine, you'll need to install:

- atlas : Automatically Tuned Linear Algebra Software (ATLAS)
- lapack : LAPACK linear algebra library

Place the compressed LAPACK source `.tgz` alongside the ATLAS source in your source directory, and execute:

```
./install.sh ATLAS
```

This will install ATLAS BLAS functions and supplement them with the LAPACK routines.

6.3.4 Build public TPLs from source

Several third party libraries (see *Third party libraries*) are necessary or recommended to build and run Exnihilo. These can be downloaded with the following links:

- openmpi : OpenMPI
- hdf5 : HDF5 scientific format library
- silo: Silo data output library
- numpy : numeric python library (optional but strongly recommended)

If installing with SCALE enabled (not needed for ADVANTG), QT is required:

- qt: large application and UI framework

To install these, simply execute:

```
./install.sh openmpi
./install.sh hdf5
./install.sh silo
./install.sh numpy
```

6.3.5 Build toolchain with MacPorts

MacPorts allows almost the entire toolchain to be downloaded and built on a Mac. You must first download the MacPorts disk image and run the included installer to get MacPorts (a package manager for the Mac platform).

To install a recent version of GCC and most of the required TPLs and tools, we have provided a script:

```
sudo xcodebuild -license
sudo ./setup_macports
```

You may want to modify/trim the list of ports in that script: it installs the Macports X11 and several other components that may be redundant for your system configuration. It also installs *iPython notebook* and other tools.

6.3.6 Build Exnihilo from source

The Exnihilo source requires three different repositories (SCALE, TriBITS, Trilinos) to build: Exnihilo, TriBITS, and Trilinos must be symlinked or moved inside the SCALE base directory. See *Getting the code* for detailed directions on obtaining and assembling these source codes.

Once the source is in place in your source directory, Exnihilo can be installed:

```
./install.sh Exnihilo {variant}
```

Here, `[variant]` is the specific set of cmake options passed to the configure. h{are:

- empty, for a typical server installation meant for users of Shift or Denovo;
- `scale-debug`, for development with SCALE support;
- `for-advantg`, if installing just for use in ADVANTG;

etc. (See the `install/codes/Exnihilo/*.cmake` and other `cmake` files in its subdirectories.)

6.3.7 Building Exnihilo from an ADVANTG distribution

The ADVANTG source distribution includes:

- a copy of the current ADVANTG source,
- a copy of the current Exnihilo source,
- a copy of the current Lava library source, and
- a skeleton copy of the SCALE build system.

It is also necessary to download the Trilinos and TriBITS source code in the same directory as these sources.

The Scale directory contains the build system infrastructure but none of the SCALE components. It has the required symlinks to `../Trilinos`, `../TriBITS`, and `../Exnihilo`.

Now you can install Exnihilo using the `for-advantg` variant located at `Exnihilo/scripts/codes/Exnihilo/for-advantg.cmake`:

```
./install.sh Exnihilo for-advantg
```

Tip: If the above installation fails for some reason, you can either modify the `for-advantg.cmake` script yourself, copy it to a new variant, etc. In these scripts, all CMake options must be set using the `CACHE` option to propagate the variables to CMake.

Once Exnihilo is installed, ADVANTG can be installed:

```
./install.sh advantg
```

and this completes the build process. See the *The install.sh script* section for details on the installer script.

For any of these steps, you of course have the option of manually creating a CMake “configure” script and running in your own build directories. Note, however, that building Exnihilo is done inside the SCALE build system, so SCALE needs to be set as the “source” directory in order to build Exnihilo.

6.4 The install.sh script

This script automates the process of “sourcing” an rc file for a particular system, setting up installation directories, and running cmake/build/install. It will execute a combination of system (with optional “system variant”) and code (with optional “code variant”). By default it will look for `rc/$(hostname)`, or if unavailable `rc/$(uname)`. It looks for install scripts or CMake configurations inside the `codes` directory, first under `codes/${code}/$(hostname)` and next under `codes/${code}`. The ability to find options under a hostname allows individual build configurations for each user of a code.

6.4.1 Install environment description

Inside the *config*`.sh` files are a number of environennt variables used by the scripts and installers that determine the installer paths, compilers, etc.

`source_base` This should be the root directory in which all the source directories are stored. For example, on most systems I use `/usr/local/src`. Inside the `src` directory I have `Exnihilo`, `openmpi-x.x.x`, `hdf5-x.x.x`, etc.

`prefix_base` This is the default parent directory of all the installed TPLs and Exnihilo. For example, if your prefix is set to `/usr/local/denovo`, you’ll end up with `.../local/denovo/openmpi`, `.../local/denovo/Exnihilo`, etc.

`build_base` This is the parent directory for the temporary builds. These files don’t need to be retained after Exnihilo is installed, so it’s often a good idea to set a directory in `/tmp` as the build root, because disk access speeds are often much faster (being on a local drive rather than a network mount for some clusters).

CC, CXX, `F90` These are the default compilers.

`mcnp_src` Currently, Lava uses this directory to build. Its immediate contents should be all the `.f90` source files used by MCNP5.

`mcnp_exe` This is the actual MCP executable, used for generating runtpe files.

`anisn_path` This is the data path to the ANISN-formatted files used by ADVANTG. It’s not necessary if SCALE is all that’s being installed.

`scale_data_path` This is the path to the SCALE data directory.

6.4.2 Example of using install.sh

Example on host `sierrajuliet`:

```
install.sh lava
```

will source `rc/sierrajuliet/base.sh`, which is a symbolic link to the `Darwin/macports.sh` resource. It will then build from the `/usr/local/src/lava` directory (the source base directory is specified in the RC file), create a build directory, and run cmake using the options specified in `codes/lava/base.cmake`. It will then start building and try to install.

Another example of installing a “code variant” (e.g. a custom debug build):

```
install.sh -t dev Exnihilo shift-release
```

will source `rc/MACHINE/dev.sh` rather than `rc/MACHINE/base.sh`; it will then run CMake based on `codes/Exnihilo/shift-release.cmake`.

There are also command line options for explicitly specifying the source, build, and install directories.

If you want to create custom source/install paths for your particular machine, simply create an `rc/HOSTNAME` directory and add scripts to it.

If the software package you're installing does not support CMake, you can create an `install_CODE.sh` file inside `codes`.

Tip: Unique builds can be specified by adding a machine variant. An example is the variant for machine *mirkwood*. Looking in the `rc/mirkwood` directory we see the system variant `debug`. To do a debug build on mirkwood in `/home/me/build` and install in `/home/me/debug` from source in `/home/me/Exnihilo`:

```
./install.sh -p /home/me/debug -b /home/me/build \
             -c /home/me/Exnihilo -t debug Exnihilo
```

Note: It is not necessary to specify the system unless you want to override the default. For example, you could use the mirkwood variant on another Mac OS X system by running:

```
./install.sh -p /home/me/debug -b /home/me/build \
             -c /home/me/Exnihilo -s mirkwood -t debug Exnihilo
```

This will use the Exnihilo base options with overrides defined in `rc/mirkwood/debug.cmake`.

6.4.3 Power user explanation

Having a script that initializes a CMake build to create a Makefile to build a piece of software is necessarily confusing. The `install.sh` script is at its essence the following script:

```
scripts_dir=$(pwd)
source rc/Darwin/macports.sh
export prefix_dir=${prefix_base}/Exnihilo
cd ${build_base}/Exnihilo
cmake -C ${scripts_dir}/codes/Exnihilo/base.sh \
    ${source_base}/Exnihilo
```

Instead of using the `-C` option to read the CMake configure file, you may define options on the command line:

```
cmake \
    -D Exnihilo_DISABLE_SCALE:BOOL=ON \
    -D SCALE_ENABLE_GeometriaLava:BOOL=ON \
    -D SCALE_ENABLE_DenovoPyKBA:BOOL=ON \
    -D SCALE_ENABLE_DenovoPySPN:BOOL=ON \
    -D CMAKE_BUILD_TYPE:STRING="Release" \
    -D ENABLE_PYTHON_WRAPPERS:BOOL=ON \
    -D ENABLE_DOCUMENTATION:BOOL=ON \
    -D SCALE_ENABLE_SWIG_EXCEPTIONS:BOOL=ON \
    -D PyKBA_ENABLE_ALL_EQUATIONS:BOOL=ON \
    -D SCALE_ENABLE_ALL_FORWARD_DEP_PACKAGES:BOOL=OFF \
    -D SCALE_ENABLE_ALL_OPTIONAL_PACKAGES:BOOL=ON \
    -D SCALE_ENABLE_SECONDARY_TESTED_CODE:BOOL=ON \
    -D BUILD_SHARED_LIBS:BOOL=ON \
    -D SCALE_ENABLE_CXX11:BOOL=ON \
    -D CMAKE_INSTALL_PREFIX:PATH="/usr/local/exnihilo" \
    Exnihilo
```

6.4.4 Command line options

-c `dir`
Manually specify the source directory name.

-b `dir`
Manually specify the build directory name.

-p `dir`
Manually specify the install directory (i.e. prefix) name.

-s `sysname`
Override the system name to load a custom RC file and cmake script.

-t `variant`
Specify a system variant, loading `rc/${sys}/${variant}.sh` as the resource file.

-m
Reuse make file if possible.

-n
Do not build or install. This option only generates the cmake configuration.

-u
Use the ctest driver to upload and run unit tests. This option is only valid when building Exnihilo or SCALE.

CHAPTER

SEVEN

WORKFLOWS

7.1 Configuration workflows

Exnihilo uses **git** for configuration management. Feature development is performed on *topic* branches. After review branches are merged into the master versions. We follow the basic tenets described in the cmake-git workflow strategy. In short, make a local *tracking* branch in your repo that diverges from the master branch:

```
$ git branch
* master
$ git checkout -b my-branch
```

Do your work and only merge back to the master when **all** unit tests pass. Feel free to commit often on the topic branch (including broken code). The objective is to only merge **correct** code back to the master. (Code on branches in a developmental state is allowed to be temporarily broken.)

Once the topic branch is completed, and all unit tests pass, do a quick code review with an Exnihilo team member. To merge the topic branch into the master, do the following:

```
$ git checkout master
$ git pull
$ git merge --no-commit my-branch
```

For details and recommendations on merging and conflict resolution, it is highly recommended that the *Merging and conflict resolution* section be consulted.

Upon successfully completing the merge, you can push to the origin:

```
$ git push origin
```

Using this basic workflow will create a clean repository in which commits are descriptively labeled, and it will be easy to walk backwards through the history:

```
        ...o      ...o
            \         \
...o----o----o----o----o  master
       /         /
   ...o      ...o
```

7.2 Design and implementation workflows

7.2.1 Design workflow

Exnihilo software design and implementation is performed using a 3-step process:

1. Documentation of numerical methods and/or algorithms to be implemented in a RNSD Technical Note:

   ```
   nemesis-note -b title
   ```

   this will generate a LaTeX technical note template.
2. Design model sketching using either UML or descriptive text; these are not formally archived.
3. Implementation with inline documentation using **doxygen**, which serves as a formal documentation of the design.

This process is highly agile and iterative. In some cases, steps 1 & 2 may be highly contracted or skipped altogether depending on the task. Step 3 can be highly iterative; thus we have found repeated updating of preliminary design sketching to be very disruptive. Informal reviews are performed after each step (see Reviews for details on reviews). Depending upon the impact/scope of the implementation, a formal review may be performed at the conclusion of the activity. The documented code is the ultimate, persistent design. All code should be self-documenting according to the coding standards described in *Coding Standards*.

7.2.2 Development workflows

Exnihilo is developed using an *agile* configure/edit/build/test cycle, often using paired-programming techniques. The *TriBITS* (Trilinos Build System) **cmake** extensions are used to control code builds, testing, and deployment. A standard development workflow is:

1. Create topic branch
2. Configure the code using cmake
3. Edit the code (perhaps as part of pair-programming or multi-programming) and corresponding unit tests
4. Build code and tests
5. Run tests Repeat until full feature is implemented
6. Review topic branch
7. Merge topic branch onto master
8. Full build + downstream tests
9. Push to server

7.3 Deployment workflow

Exnihilo is developed using a test-driven, continuous-integration process. Thus, each version of master can be successfully deployed as a release because it successfully runs all unit and acceptance tests.

7.4 Reviews

The decision upon whether to perform a *formal* or *informal* review is up to the Exnihilo team. Generally, factors such as complexity, importance, and difficulty determine whether a formal or informal review needs to be performed. The objective of reviews is to identify issues as close to implementation as possible; this dramatically reduces defect resolution times and minimizes adverse impacts across the project. The majority of reviews in Exnihilo are *informal*.

7.4.1 Formal reviews

Formal reviews are conducted by a review panel. The size, scope, and members of the panel are determined by the item being reviewed. The following table gives recommended numbers and team composition for different types of reviews.

Reviewed Item	Panel Size	Team Composition
Code walk-through	1–2	team members only
Code review	2–3	may include an external member
Design review	2–3	should include an external member
Requirements review	2–3	should include an external member, preferably a project stakeholder
Methods review	2–3	should include an appropriate subject matter expert
User and developer manuals	2–3	should include an external member, preferably a user

This is not an exhaustive list as many activities outside of these areas may require review. Use this information as a guide for such activities.

The requirements for formal review are:

- review minutes should be documented using **iPython Notebook** and archived in the Exnihilo documents repository (`ssh://angband/repos/git/documents.git`) under `reviews`.
- reviews must be moderated and have a time-limit
- issues should be discussed, not solutions

Review minute documentation need not be exhaustive or comprehensive. Important issues and outstanding action items should be noted.

7.4.2 Informal reviews

Informal reviews take nearly any form. The most effective format is a *walk-through*. This type of review has the creator of an artifact review the item with another person. The other person is usually a team member. No formal documentation or archiving is required for informal reviews.

CHAPTER

EIGHT

CODING STANDARDS

This chapter gives the coding standards used in Exnihilo. Most of the software in Exnihilo is be written in C++. The power of C++ can be abused easily, resulting in code that is difficult to understand and maintain. This document gives the practices that must be followed on the Exnihilo project for all new code that is written. The intent is not to be onerous but to ensure that the code is solid and maintainable. As of June 1, 2014, the Exnihilo code base will be integrating C++11 constructs into the code base. Accepted C++11 practices are described in C++11 Usage.

External libraries that Exnihilo uses do not have to meet these requirements, although we encourage external developers to follow these practices. For any code that Exnihilo takes ownership, the project will decide on a case-by-case basis on any changes.

8.1 Organizing and writing code

This section deals mainly with how code should be organized and written in terms of files, file names, code formatting, and documentation.

8.1.1 File names

Each class is defined within its own set of files, as summarized in the following table:

Table 8.1: Exnihilo TPLs

File	Description
`A.hh`	*Header file*. Contains definition of class `A`. It may also contain member function definitions, although preferably, function definitions should be in one or more of the files `A.cc`, `A.i.hh`, and `A.t.hh`.
`A.cc`	*Implementation file*. If `A` is non-templated, then contains non-templated member function definitions of `A`. If `A` is templated, contains member function definitions of specializations of `A`.
`A.t.hh`	*Template implementation file*. May be used if `A` is a templated class, or if `A` contains templated member functions. In these cases, contains the corresponding function definitions which will be explicitly instantiated by `A.t.cc`.
`A.i.hh`	*Implementation file for member functions*. This file should **always** be included at the bottom of `A.hh`. If `A` is templated, or has templated member functions, then this file can be used for an automatic instantiation model. For non-templated entities, this file may contain `inline` function definitions.
`A.pt.cc`	*Instantiation file*. Used for explicit template instantiation of the definitions in `A.t.hh`. For specific template arguments, contains instantiations of `A` or its templated member functions.
`test/tstA.cc`	*Unit test file*. The unit test for `A`. Other files may also be used by the unit test.

Note: Multiple classes should not be defined in a single set of files, except in very special circumstances where the classes are closely related (for example, nested classes, or an iterator class for a container).

8.1.2 Generating files

The Exnihilo *Development Environment* installs a `template-gen` tool that creates templates for various file types. To generate files for a new templated class called `Foo`, you might do the following:

```
$ template-gen Foo.{hh,t.hh}
>>> Sucessfully created Foo.hh
>>> Sucessfully created Foo.t.hh
$ template-gen test/tstFoo.cc
>>> Sucessfully created test/tstFoo.cc
```

This will create the class header, the template definition file, and a unit test template.

8.1.3 Template model

In Exnihilo, there are two models that are used for template instantiation, based on usage:

Automatic This is for templated classes for which it is unlikely that the template arguments are known beforehand. Examples are containers and smart pointers (which can contain or point to any data type). These classes include their function definitions within their header file (`*.hh`, either explicitly or via a `*.i.hh` file. Hopefully, to avoid code bloat and excessive compile times, these classes are small.

Explicit This is for templated classes for which the range of template arguments is known. An example is a finite-volume class that is templated on the mesh type (the number of mesh types one typically requires is known and fairly small). Here, the function definitions (`*.t.hh`) are included by the template instantiation file (`*.t.cc`). The instantiation file instantiates the class for each template argument that is desired.

Note that the STL follows the automatic instantiation model.

8.1.4 Write unit tests

The author of a class must write a unit test for that class. The unit test not only tests the functionality of the class, but also helps serve as an example for how to use the class. Some authors actually prefer to write the test before the class. See File Names for where the unit test should reside.

8.1.5 Syntax names and code formatting

Rules for names and code formatting can be onerous. We believe many such rules are based more on personal preference and do not significantly add value to a code's readability. If too many rules are specified, teams tend to ignore **all** the rules, or become unhappy with enforcement. The intent here is to have code that is quickly understandable by anyone on the Exnihilo team. But individual team members may not find each other's style "pretty."

Hint: With regards to code style, the most important thing is to be neat and consistent. Your code will be read by other team members, so readability is important. Code that is sloppy will not be accepted into the head version of the code. Try to follow existing source code conventions when possible.

Important: The most important stylistic guideline imposed by the Exnihilo code team (without exception) is an 80 column limit on statement lines. Please limit all code statements to no more than 80 columns.

Consequently, we have narrowed down the list of rules to those we believe are the most important (in addition to the inviolate 80 column limit mentioned above):

1. Indent your code as follows:
   - Do not indent curly braces relative to their control statements.
   - Do not indent `namespace` blocks.
   - Indent `public`, `private`, `protected` statements two spaces relative to their class' braces.
   - For other code blocks, indent four spaces relative to their enclosing braces.
   - Comments on their own line should be indented to the same level as the code they are commenting.
2. Be consistent in spacing, bracket placement, formatting, etc. If modifying someone else's code, respect and attempt to follow their style (otherwise, you are introducing inconsistency).
3. Separate words and acronyms within a name with an underscore (if you **must** use CamelCase, be consistent throughtout the class/scope).
4. Prefer complete words or acronyms in names. For example, use `is_anal_retentive` instead of `is_anl_ret`.
5. Distinguish variable and function names from user-defined type names as follows: Begin all words in type names with a capital letter. Begin all words in variable and function names with a lowercase letter.
6. One should not have to search outside of a file, or parse an entire function definition, to determine the origin of a name used within that file (ideally, apply this down to function definitions). This rule implies the following:
   - Use `d_` for members of a class. Use `b_` for members of a base class. Define the private `typename Base` in a derived class as a reference for the `Base` class, ie:

```
class A
{
  public:
    A(int a, int b);

  private:
    // Base class members that do something important (notice that
    // these members are commented?).
    int b_a, b_b;
};

class B : public A
{
    typedef A Base;
  public:

    explicit B(int x);

  private:
    // This is an important datum.
    int d_important_datum;
};
```

     This allows constructors and virtual functions to clearly and easily reference the parent class:

```
B::B(int x)
    : Base(x + 1, x + 2)
    , d_important_datum(x)
{
        /* * */
    }
```

- Even within an implementation file, avoid `using namespace`. One exception here is `using namespace std`, as long as you are using very common stuff from `std` (`cout`, `endl`, `vector`,...). It is still preferable that you explicitly specify which names you're using (e.g., `using std::cout`).

7. Limit the body of a function definition to approximately one page in length.
8. Declare only one variable per line, including within function definitions:

```
void blorp(int i,
           int j)
{
    // preparing to blorp
    // ...
}
```

9. In general, do only one operation per line. For example, avoid:

```
a = b = c = d = 0;
```

however, this is acceptable:

```
int a = 0, b = 0, c = 0, d = 0;
```

8.1.6 Code comments and documentation

Code must be reasonably commented and documented using **Doxygen**. The Exnihilo templates provide code-blocks for **Doxygen** comments. Also, the Exnihilo development environment provides **emacs** and **vim** define macros and menus for documenting the code as described in *Development Environment*. Our preference is to put one-line comments for each member function in the header file and its documentation in the implementation file.

8.2 Use of language features

This section gives rules on language features to use and to avoid. Items specifically related to C++11 can be found in C++11 Usage.

8.2.1 Use namespaces

All code should be contained within a **namespace**. The name of the **namespace** is generally (with exceptions) taken from the package containing the code; although, it is acceptable to use subpackage namespaces when these make logical design sense.

8.2.2 Enforce const-correctness

When declaring a variable or function, use the `const` qualifier whenever possible. If unsure when declaring the variable, then go ahead and add the `const` qualifier. You can always remove the qualifier later. We discourage strongly the practice of adding const-correctness after major features have been implemented.

Denovo may have to use external libraries that do not enforce const-correctness. Such libraries must be wrapped in the equivalent functionality that enforces const-correctness. Otherwise, if wrapping is not done, the external library potentially could force all of Denovo to abandon const-correctness. Wrapping has other benefits, such as that the external library can be swapped out.

8.2.3 Design-by-Contract (TM)

The Design-by-Contract (DBC) macros defined in are defined in `harness/DBC.hh`. The following examples shows how DBC can be used:

```
#include "harness/DBC.hh" // defines Require, Check, Ensure, ...

double pressure(double temperature,
                double density)
{
    Require (temperature >= 0.0); // use Require() to check input values
    Require (density >= 0.0);

    double p; // pressure that is returned

    // ... code that computes initial guess for p ...

    Check (p >= 0.0); // use Check() for intermediate calculations

    // ... code that computes final value of p ...

    Ensure (p >= 0.0); // use Ensure() for final values

    return p;
}
```

8.2.4 Data Hiding

Aside from pure data structures (i.e., a `struct`), class member data should be accessed only through member functions. There are several suggestions for accessors. Consider the following example with accessors to large data structures; presumably, the `CCF` class contains a large amount of data. Care must be taken that `Solution` is not destroyed while handles it has returned are still available.:

```
class Solution
{
    // >>> DATA

    typedef std::vector<double> CCF; // cell-centered field
    CCF d_pressure;
    CCF d_density;

  public:

    // >>> ACCESSORS

    CCF &pressure() { return d_pressure; }
    const CCF &pressure() const { return d_pressure; }

    CCF &density() { return d_density; }
    const CCF &density() const { return d_density; }

    // >>> ITERATORS
    CCF::iterator begin_pressure() { return d_pressure.begin() }
    CCF::iterator end_pressure() { return d_pressure.end() }

    // ... etc ...
};
```

A memory-safe solution is the use the `RCF` (Reference Counted Field) class; however, this is may not be optimal for all use cases,:

```
#include "utils/RCF.hh"
class Solution
{
    // >>> DATA

    typedef denovo::RCF< std::vector<double> > CCF; // cell-centered field
    CCF d_pressure;
    CCF d_density;

  public:

    // >>> ACCESSORS

    CCF pressure() { return d_pressure; }

    CCF density() { return d_density; }

    // >>> ITERATORS

    CCF::iterator begin_pressure() { return d_pressure.begin() }
    CCF::iterator end_pressure() { return d_pressure.end() }

    // ... etc ...
};
```

The data pointed to by any `RCF` will not go out of scope until the last one is destroyed. Also, copies are cheap because only the underlying reference is copies, not the data itself. Of course, this allows any owners of these fields to modify the underlying fields.

With regards to data hiding, optimization, ease of use, and safety must dictate design. These constraints are often conflicting. The following guidelines are helpful to keep in mind:

- If the class is a data container (for example, a container of cell-centered fluxes on a mesh), then data container semantics may be used to access the underlying data. These semantics include `operator[]`, `operator()`, and iterator access.
- If the member data is a large data structure, then for efficiency, handles to that data may be returned. The handles may be in the form of iterators or references, as in the first example above.
- Otherwise, *get*/*set* semantics should be used. The @code{get} function should not return a non-const handle to the data.

8.2.5 The `std` namespace

For example, use `#include <cmath>` instead of `#include <math.h>`. In general, do not use the `.h` system header files, which pollutes the global namespace. The resulting code will have the following constructions:

```
#include <cmath>
#include <iostream>
```

```
// ... stuff
x = std::sqrt(y);
std::cout << x << std::endl;
```

8.2.6 `using` statements

Do not place `using` statements where they might pollute the global namespace. Unless within the scope of an `inline` function, `using` statements should not be placed within header files (`*.hh`). Even within implementation files, preferably `using` statements should appear only within function scope.

Consider a `A.hh` file that contains the following:

```
#include <iostream>

namespace using_abuse
{

  using namespace std; // Not here!!!

  class A
  {
    public:
      void print_something() { cout << "I am lazy.\n"; }
  };

} // end of namespace using_abuse
```

Now, consider a translation unit that uses `A.hh`:

```
#include "A.hh"

int main()
{
    // The following using statement is OK, because it's in an implementation
    // file.  Unforunately, it asks for using_abuse, but got std too!!!
    using namespace using_abuse;

    A a; // using_abuse::A is OK too
    a.print_something();

    cout << "Where did cout come from???\n"; // Answer: via using_abuse.
}
```

You might argue that `namespace using_abuse` is "yours," and you're free to pollute it all you like. However, someone else might have to maintain your code in the future. Also, placing `using` statements with header files may affect those who want to use your class, as illustrated above.

8.3 Things to avoid

The following is a list of things to avoid. It is not comprehensive, and like everything, there are times when "rules" need to be broken.

8.3.1 Circular dependencies

Circular depencies arise from two-way associations. Denovo subscribes to the principles of *levelized design* (sometimes referred to as *acyclic* design). This is critical to the unit-testing and continuous-integration lifecycle models in Exnihilo. Thus, circular dependencies are **not** allowed. Additionally, there is never a reason to use circular dependencies (an *association* class can be used to decouple two-way associations). A simple example of a circular dependency is shown below:

```
class B; // forward declaration

class A
{
    int do_b(const B &b); // refers to class B
};

class B
{
    int do_a(const A &a); // refers to class A
};
```

Because both classes A and B refer to one another, they cannot be tested independently. Circular dependencies also create build-system nightmares.

8.3.2 Friendship

There are specialized cases where friend is useful (such as an iterator class for a container class), but generally, the use of friend is strongly discouraged. The use of friend violates data hiding, which was covered in the previous section. Remember, “in C++ friendship, much as in life, is often more trouble than its worth.”

8.3.3 Macro functions

Macro functions are not type-safe and should generally be avoided. This is not to imply that macros should **never** by used. We use them for our DBC implementation and conditional compiling. Simply be judicious when using macros.

8.3.4 Raw pointers

The use of raw pointers (for example, double *x) can be a major source of bugs. Often, the use of raw pointers can be avoided by substituting one of following techniques:

- Use a *shared pointer* instead (either std::shared_ptr or std::unique_ptr).
- Use a reference.

There are situations where using a raw pointer cannot be avoided. For example, raw pointers cannot be avoided when communicating with other languages, such as Fortran or C. In other cases, their use should be encapsulated. For example, container classes often use a pointer for their underlying data storage and may define its iterator type as pointer via a typedef. However, the pointer implementation in this case is encapsulated from the user of the container class. It is very important to avoid the following constructs:

```
class Foo_Factory
{
  private:
    Foo *d_foo;

  public:
    void build()
    {
        // build foo
        d_foo = new Foo();
        // ...
    }

    Foo* get_Foo() const { return d_foo; }
};
```

The issue is, “who ultimately deletes the Foo here?”. By returning a `SP<Foo>` this question is null. The last remaining owner will delete the Foo.

Finally, pointers-to-functions are a relic of C and can be avoided through the use of virtual functions.

8.4 C++11 usage

C++11 has many useful features; however, in Exnihilo we recommend a subset of features to use. The best place to discover the accepted use-patterns of C++11 features is to peruse the examples in `Nemesis/cxx11`. One thing to be wary of is that many compilers outside of **gcc** do not have good support for many C++11 features. And, **gcc** before 4.7.1 has some limitations as well. The following is a basic list of C++11 best practices:

- Keep the `auto` keyword usage *local* (ie. at function-scope). Avoid using `auto` in argument lists.
- Use `nullptr` to initialize raw pointers, *when raw pointers are absolutely necessary*.
- Use `unordered_map` and other new standard library containers; however, beware that the `emplace` is not supported by several compilers (see `tstCXX11_Std.cc`).

  Note on Smart Pointers We will be making the conversion from native Exnihilo smart pointers (`SP` class) to the new C++11 standard smart pointer, `shared_ptr`. This is a transition, and `shared_ptr` should be avoided except inside of local class/scope internals at this point.
- Respect the Rule of 5 (formerly known as the *Rule of 3*).

CHAPTER

NINE

TESTING

Exnihilo contains two primary types of unit tests based on the two coding languages used.

9.1 C++ unit tests

Exnihilo uses the Google Test framework for most of its unit tests. (Some tests in the repository use an obsolete test harness.)

9.1.1 Creating a unit test

The Nemesis environment (*Development Environment*) installs a handy script for generating a template for new tests inside a test directory:

```
$ template-gen -n shift tstBlah.cc
Successfully created tstBlah.cc
```

Here, the `-n` option is followed by the namespace. This creates a new google test file. If you don't have the environment scripts installed, simply create a new `.cc` file with the following line among the includes:

```
#include "Nemesis/gtest/nemesis_gtest.hh"
```

This includes the Google Test header file, declares additional Nemesis macros, and injects the custom `main()` function needed to run all the tests.

9.1.2 Adding the test

To build and automatically run the test, you will need to modify the `CMakeLists.txt` file inside the test directory. This file must include the following setup line:

```
INCLUDE(NemesisTest)
```

To build and run the test, add the simple macro directive:

```
ADD_NEMESIS_TEST(tstBlah.cc  NP 1 2)
```

The `NP 1 2` option indicates the number of processors to allow to run the test. The full list of options (including setting environmental variables, linking in libraries, adjusting the timeout, and suppressing the running of the test through CTest) can be viewed inside the `Exnihilo/packages/Nemesis/cmake/NemesisTest.cmake` file.

9.1.3 Testing functionality

If you've generated the test file from the Nemesis environment utlity, you'll see a file header, the command to include the gtest harness, and an empty "test fixture", along with two example testing blocks. Each `TEST_F` or `TEST` command generates a distinct subtest that appears in the output. This is a good way of grouping related tests. Each test contains testing macros that will print nothing when they succeed but will provide detailed error descriptions when they fail.

```
TEST(Math, simple)
{
    int a = 2;
    EXPECT_EQ(4, 2 + 2);
    EXPECT_GT(5, 2 * 2);
    EXPECT_TRUE(is_integer(a));
    EXPECT_FALSE(is_float(a));
}
```

A complete list of the test macros is available in the Google Test primer and the advanced guide.

9.1.4 Additional functionality

The test harness code defined in `nemesis_gtest.hh` includes additional functionality not written by Google. It has a custom harness that handles MPI initializing and finalizing (using the Nemesis comm package), and it uses the Nemesis release metadata to print the Exnihilo and Scale git repository versions.

Additional macros that we define are mostly for comparing floating point values or containers of values:

`EXPECT_SOFTEQ`(expected, actual, rel_error)

Check for "soft equivalence" (relative error except for numbers near zero) between two values.

Parameters

- **expected** – expected value
- **actual** – actual value
- **rel_error** – numerical tolerance

`EXPECT_SOFT_EQ`(expected, actual)

Check for "soft equivalence" with a relative error of 1.e-12.

`EXPECT_VEC_EQ`(expected, actual)

Check for equality between two contiguous containers of simple types (int, unsigned int, float, double, char, const char*, std::string). Each container can either be a fixed array such as:

```
int reference[] = {1,2,3};
```

or a contiguous container (std::vector, Nemesis view field, etc).

If the two container sizes are unequal, the test prints their sizes and returns a failure message. If the sizes are equal, it will do an element-by-element comparison. Any values (up to the first 40) that do not compare equal are printed along with their index:

```
tstVecEq.cc:87: Failure
Values in: actual
 Expected: expected
2 of 5 elements differ:
i         expected            actual
0                1                 2
3                4                 5
```

Parameters

- **expected** – expected value
- **actual** – actual value

EXPECT_VEC_SOFTEQ(expected, actual, rel_error)
Check for "soft equivalence" (relative error except for numbers near zero) between two vectors of floating point values.

Parameters

- **expected** – expected value
- **actual** – actual value
- **rel_error** – numerical tolerance

EXPECT_VEC_SOFTEQ(expected, actual)
Check for vector soft equivalence with a relative error of 1.e-12.

We also provide a function that's useful for printing "regression" values of integers or floats:

PRINT_EXPECTED(data)
Print a formatted array and initializer list for the given data. For example:

```
std::vector<int> vec = {1,2,3,4,5};

PRINT_EXPECTED(vec);
```

will print:

```
const int expected_vec[] = {1, 2, 3, 4, 5, };
```

The resulting values can then be copy-pasted to the unit test and used in the expression:

```
EXPECT_VEC_EQ(expected_vec, vec);
```

9.1.5 Command line options

The Google test harness provides a number of useful command line options that can be viewed by passing the `--help` argument to the command line:

Each individual test block can be executed separately by running the test with the `--gtest_filter=TestCase.testname` flag.

9.1.6 Other features worth looking into

- Disabling performance tests by prepending `DISABLED_` to a test name.
- Using the `ASSERT_` macro to exit a test function when all further tests are known to fail (or cause crashes) if a condition is not met.
- Adding `EXPECT_THROW({func}, nemesis::assertion)` to ensure that DBC checks work.

9.2 Python unit tests

Exnihilo installs the `exnihilotest` package and the `denovo_unittest` module, which both wrap and extend the Python unittest package. Combined, they give the ability to do more advanced testing, they integrate into TriBITS, and they can run correctly and automatically with MPI.

9.2.1 Creating a unit test

Create a new `.py` file, either with the Nemesis `nemesis-python` command or with the function from srjutils, `template-gen tstWhatever.py`.

```
$ template-gen test_mymodule.py
Sucessfully created test_mymodule.py
```

The file (with dividers omitted for clarity) should look like:

```
1  from __future__ import (division, absolute_import, print_function, )
2  import omnibus.testing as unittest
3
4  if __name__ == '__main__':
5      unittest.main()
```

Line 1 contains a standard directive to make Python 2.7 behave more like Python 3: dividing integers yields a floating point number (use the `//` operator for C-like integer division); module imports are by default absolute rather than relative; and the `print` command acts like a function.

The second line acts as a stand-in for the standard Python `unittest` module. If testing a standalone Python package like `swordproc`, replace `omnibus.testing` with `exnihilotest`; if testing SWIG-wrapped code that might run in parallel (e.g. in the `pykba` or `pyshift` packages), then replace `omnibus.testing` with `denovo_unittest`.

Lines 4 and 5 will execute all the unit tests in the file when the script is run.

9.2.2 Adding to a unit test

Unit tests inside the file are created by adding a new class that begins with `Test`, inherits from `unittest.TestCase`, and has at least one method that begins with `test_`:

```
class TestSomething(unittest.TestCase):
    def test_blah(self):
        self.assertEqual(4, 2+2)
```

Inside the methods are assertions that check for equality between expected and actual results. Each test function is run independently, so if one assertion fails, separate tests may still be run. For an extensive look at the capabilities of the package, see the full unittest documentation.

9.2.3 Additional functionality

The `exnihilotest` package defines several useful macros not available in the standard `unittest` package.

`TestCase.`**`assertDataEqual`**(*self*, *expected*, *actual*[, *eps=1.e-14*])

Check all elements recursively in a container. This works on dictionaries, numpy arrays, lists of lists, named tuples, etc.

Parameters **eps** (*float*) – "Soft equivalence" tolerance parameter.

`TestCase.`**`assertTextEqual`**(*self*, *expected*, *actual*[, *eps=1.e-14*])
Compare blocks of text which may have numbers inside it.

Parameters **eps** (*float*) – "Soft equivalence" tolerance parameter.

`TestCase.`**`assertSoftEquiv`**(*self*, *expected*, *actual*[, *eps=1.e-14*])
Check soft equivalence on two floats.

Parameters **eps** (*float*) – "Soft equivalence" tolerance parameter.

`TestCase.`**`assertXmlFilesEqual`**(*self*, *expected_f*, *actual_f*, *eps=1.e-14*)
Compare the structure and contents of two XML files for equality.

Parameters

- **expected_f** (*str*) – Expected file object or file path.
- **actual_f** (*str*) – Actual file object or file path.
- **eps** (*float*) – "Soft equivalence" tolerance parameter.

The `denovo_unittest` package prints warnings emitted by `NEMESIS_WARN` at the end of every test that it runs. When loaded, it prints the current version of Exnihilo and SCALE.

CHAPTER

TEN

DEVELOPMENT ENVIRONMENT

Exnihilo provides templates and functions for developing code in `Exnihilo/environment`.

10.1 Setting up the Exnihilo development environment

We provide **emacs** and **vim** environments for editing source code. Also, a set of code preparation templates, along with LaTeX document styles are provided. To install the environment simply run the following script in the `Exnihilo/environment` directory:

```
sh ./install.sh /data/env
```

Here, the `/data/env` path should be where you want the environment to be installed. The following output is produced:

```
>>> Making /data/env
>>> Installing bibtex in /data/env/bibtex
>>> Installing emacs in /data/env/emacs
>>> Installing latex in /data/env/latex
>>> Installing etc/templates in /data/env/etc/templates
>>> Installing tools in /data/env/tools
>>> Installing visit in /data/env/visit
>>> Installing python in /data/env/python
>>> Installing bin in /data/env/bin

========================================================================
Congratulations, you're almost finished the nemesis environment install!

For complete operation you should set the following paths and
environment variables (depending upon your shell):

 - Add /data/env/bin to PATH
 - Add /data/env/latex to TEXINPUTS
   e.g. echo $TEXINPUTS
        .:/data/env/latex:
 - Add /data/env/bibtex to BSTINPUTS
   e.g. echo $BSTINPUTS
        .:/data/env/bibtex:
 - Copy /data/env/visit/* to ~/.visit

To use the nemesis GNU Emacs or VIM environments see
 - environment/README.rst
========================================================================
```

The script copies several recommended VisIT color palettes; to use them, copy `install/visit/*` into `~/.visit`.

10.2 Command line tools

Several useful development tools are installed into `environment/tools`. A few of the more useful ones are described here. Many of these tools use a small Python framework that allows files to be recursively modified based on their extensions.

10.2.1 auto-annex

Intelligently add or git-annex untracked files.

```
usage: auto-annex [path [path ...]]
```

Each path (either a file or directory) will be examined. If it has not yet been added to the Git repository, a heuristic algorithm will decide whether to annex it or to add it as a regular git file:

- If a binary file [1] has a size greater than 5 KiB, it is annexed.
- If an image file [2] has a size greater than 25 KiB, it is annexed.
- If an output file [3] has a size greater than 100 KiB, it is annexed.
- If any other file has a size greater than 2 MB, it is annexed.
- Any file less than these size thresholds is added as a regular git file.

10.2.2 check-cols

Determine whether source files are violating the Nemesis 80-column guideline.

```
usage: check-cols [-h] [-r] [-x EXTENSIONS] path [path ...]

Check for satisfying the 80-column limit. If violating files were found, a
file "violators" will be written containing their paths.

positional arguments:
  path                  Files/dirs to process

optional arguments:
  -h, --help            show this help message and exit
  -r                    Recursive
  -x EXTENSIONS, --extensions EXTENSIONS
                        Comma-separated extentions to use when doing recursive
```

[1] Binary files are determined by checking the extension against `h5 sh5 silo ampx dat bin bmg h5m sat cub dnv 37 bin docx pptx xlsx ppt doc xls session gui`.
[2] Image files have en extension `png pdf jpg psd svg`.
[3] Output file names either have extensions of `mcnpo out log plt`, have paths that end with `out.xml` or `output.xml`, or look like MCNP `mctal` or `meshtal` or `outp` files.

10.2.3 fix-comment-paths

Modify the comment blocks at the beginning and end of Nemesis-style files to ensure that the declared file path is consistent with the actual file path.

```
usage: fix-comment-paths [-h] [-r] [-x EXTENSIONS] path [path ...]

Update file paths in the comment headers and footers of Nemesis-style files.

positional arguments:
  path                  Files/dirs to process

optional arguments:
  -h, --help            show this help message and exit
  -r                    Recursive
  -x EXTENSIONS, --extensions EXTENSIONS
                        Comma-separated extentions to use when doing recursive
```

10.2.4 make-skeleton-scale

Create a "skeleton SCALE" source directory used to build Exnihilo without any of the SCALE source code. This is primarily used for distributions of ADVANTG, where the SCALE cross section processing code is not used.

```
usage: make-skeleton-scale SCALEDIR DEST.tgz
```

10.2.5 update-scale-data

Update the SCALE data directory to the latest version of SCALE data. Your `${DATA}` environment variable must be set for this to work. The first time you run the tool, you must run `mkdir ${DATA}` to create the initial empty directory on your machine.

```
usage: update-scale-data

Update the SCALE data directory (using the DATA environment variable)
from the /projects/scale/scale_dev_data directory on remus.ornl.gov.
```

Note: In addition to copying the SCALE data itself, this tool saves a copy of the Subversion repository data (version number and date of change), which is processed by Exnihilo and saved into Omnibus output files for later verification.

10.3 Text editor environments

As previously mentioned, both **emacs** and **vim** editing environments are provided that correspond to the Exnihilo indentation style, along with many other useful editing features.

10.3.1 EMACS Development Environment

To use the nemesis GNU **emacs** environment add the following to your `.emacs` or `.emacs.d/init.el` file:

```
(setq nemesis-dir "/data/env/emacs")
(add-to-list 'load-path nemesis-dir)
(load-library "nemesis")
```

Other useful emacs packages/modes include

- **doxymacs**: for **doxygen** markup in source code
- **cmake**: for **cmake** editing
- **auctex**: for LaTeX editing
- **ecb**: the **emacs** code-browser extensions
- **auto-complete**: auto-completion of symbols (code)

10.3.2 VIM Development Environment

These are **vim** format commands for compliance with the Exnihilo coding guide. Simply add the `ftplugin` files to your `~/.vim/ftplugin` , and append `vimrc` to your existing `~/.vimrc`.

CHAPTER

ELEVEN

USEFUL TOOLS

The Exnihilo developers tend to use a common group of tools, including git for versioning, Emacs and MacVim for file editing, etc. This document provides more details on both the standard tool set as well as additional tools in use.

11.1 Git

Git at first is a daunting tool, but a little experience and guidance makes it a powerful ally in the quest for a maintainable code system.

11.1.1 Git configuration

Git supports a configuration file at `~/.gitconfig`. A recommended default is:

```
[core]
    excludesfile = ~/.gitignore
    pager = less -r
    autocrlf = input
    whitespace = trailing-space,space-before-tab
[apply]
    whitespace = fix
[color]
    ui = auto
[alias]
    autoannex = !auto-annex
    st = status -s
    co = checkout
    dc = diff --cached --ignore-space-change
    mnc = merge --no-commit --no-ff -s recursive -Xignore-space-change
    comall = commit --all
[branch]
    autosetuprebase = always
[rebase]
    autosquash = true
[push]
    default = upstream
[rerere]
    enabled = true
[merge]
    autosetuprebase = always
[user]
    email = johnsonsr@ornl.gov
    name = Seth R Johnson
```

Don’t forget to change the email and name!

The init file points to a global `.gitignore` file that hides temporary files from Git, preventing them from accidentally being included in a commit. A recommended default is:

```
*.aux
*.pyc
*.out
*.log
*.bbl
*.blg
*.synctex.gz
*.nav
*.snm
*.toc
*.swp
*.swo
.DS_Store
visitlog.py
*~
runtp*
srctp*
.ipynb_checkpoints
.cecache*
```

11.1.2 Merging and conflict resolution

To view all the changes your branch has made since splitting off from master, use the command:

```
$ git diff $(git merge-base origin/master HEAD)
```

To merge the topic branch into the master, do the following:

```
$ git checkout master
$ git reset --hard origin/master
$ git merge --no-commit my-branch
```

Resolve any conflicts and run all (downstream) tests to ensure that the merged branch does not cause any failures. Conflicts may be either *explicit*, where the master and the topic have both modified a line of code, or they may be *implicit*, where a new file on the master relies on old behavior that the topic branch changes.

Caution: Implicit conflicts are subtle and cannot be caught by Git’s merge resolution mechanisms. Consider the following case:

```
   ...N----o (other developer)
            \
...o----o----o----o---M **master**
         \           /
          o----R----o *(your topic)*
```

Commit `R` in your topic branch renames a class MyClass to MyRenamedClass, and some other developer on their branch creates a new class at commit `N` that references MyClass. The other developer merges into the master branch and everything works. You finish up your topic branch, and everything works. Because your topic branch and the other developer’s branch didn’t touch the same files, merging into master at `M` compile individually, and no merge conflicts will be generated. However, because the new class the other developer created references MyClass instead of MyRenamedClass, the merge will fail to compile! This is why it’s important to always build and test before pushing.

Upon successfully completing the merge, you can push to the origin:

```
$ git push origin
```

Using this basic workflow will create a clean repository in which commits are descriptively labeled, and it will be easy to walk backwards through the history.

11.1.3 Advanced merging

When resolving a long-running branch, where significant restructuring to code has been performed, it's often easier to complete the conflict resolution in multiple stages.

Tip: When merging a topic branch that contains a significant code refactor, it takes time to resolve conflicts. The more time it takes, the more likely it is that another developer pushes changes onto the master branch, and the more frustrating it will be to redo the merge.

To mitigate this frustration, our git server supports the ability to "lock" the repository while the merge is taking place; talk to one of the Exnihilo developers for details on how and when to do this, and it will reduce the likelihood of your having to use the advanced techniques laid out in this section.

The first step is to make sure that the "rerere" option given in *Git configuration* is set; this allows Git to remember how you resolved your merge in case you have to perform the merge again. Then, run `git merge --no-commit my-branch`, resolving explicit merge conflicts as needed. Run `git diff` to make sure that no conflict resolution markers are left in the code, then run `git commit --all`.

The next major step is to resolve any *implicit* conflicts. Start building the merged code. If it builds and the tests pass, you can push. If there are build errors or test failures from the merge, fix them, mark the changes to be added with `git add -A :/`, and make a commit with `git commit -m "Merge fixes"`. This creates a checkpoint of all the merge conflicts that Git won't be able to automatically resolve.

If the master has *not* been updated since you started your merge (run `git fetch` to check), you can run the following code to collapse your rolls both the explicit and implicit conflicts into the single merge commit:

```
$ git reset --soft HEAD^
$ git commit --amend
```

Then `git push` and you're done.

However, if someone *has* pushed to master while you were doing your merge, you have a problem:

```
       ...o---o  *(unsuspecting developer's topic)*
               \
...o----o---o---o  **origin/master**
             \
              M---N  **master** (your original merge in progress)
             /
    ...-o---o *(your topic)*
```

In this case your original merge resolutions are in `M`, the additional merge fixes are in `N`, and your `master` has diverged from `origin/master`. If you try to push, you'll get a message saying your push was rejected.

Warning: If this happens, *do not* merge origin/master into your merge commit, and also *do not* merge `M'` into origin/master!

But fear not! There is an easy solution. You can recreate your merge on top of the updated `origin/master` because you separated "implicit" merge conflicts into a separate commit and because you have the "rerere" option enabled. First, copy the git hash so that you can later refer to your original merge attempt:

```
$ git log -1 --oneline
abc123 Merge fixes
```

To rebase your merge, use the `--preserve-merges` option:

```
$ git fetch origin
$ git rebase -p origin/master
```

which rebases your merge commit. You'll get a message about "recorded resolutions" being replayed; you should have to commit to acknowledge that they were applied correctly, and then the rebase should complete. You should end up with:

```
       ...o---o  *(unsuspecting developer's topic)*
               \
...o----o---o---o---M'---N'   **new master**
                   /
          ...-o---o *(your topic)*
```

Then you can "roll" your `N'` into `M'`. (Double-check that no new conflicts have been created, of course.) To do a final check that you didn't lose anything in this rebase, you can refer to the earlier git hash you saved and run a git diff:

```
$ git diff --stat abc123
```

The only differences should be those the unsuspecting developer introduced in their branch.

Tip: If you didn't save your git hash earlier, you can recover it by looking through the results of `git reflog` and finding the commit named "Merge fixes".

Once everything is checked, you are of course free to push.

11.2 Git annex

Git-annex is an add-on to Git designed for safely syncing large files between repositories. Many codes output binary files that measure in GB or more. Git is slow to handle such files, and every user who pulls a copy of a research repository must also pull all those large files. Because of Git's nature, it's also nearly impossible to delete one of those files when it's superseded by a newly regenerated version.

Git-annex solves these problems by moving large files into a hidden directory inside your git repository and using `rsync` under the hood to copy the files between computers. It maintains an awareness of where your files are located (through a hidden git branch) and will prevent them from being accidentally deleted.

For example, when git-annex is configured locally and on the server, to add some large research result alongside some smaller text documents, you might do the following:

```
# First, add large files to the annex. This moves them to
# .git/annex/objects and creates symlinks in your directory.
git annex add *.h5
git add .
git commit -m "Added some files"
```

Now these large files are stored securely on your computer. Calling `git rm` will not delete the original files; you'd have to run `git annex drop` to remove them.

11.2.1 Installation

On the Mac, the easiest way to install is to download the OS X binary. After you copy the app to `/Applications`, you can symbolically link the following important files to `/usr/local/bin` or some other convenient location:

```
cd /Applications/git-annex.app/Contents/MacOS/bundle/
ln -s git-annex git-annex-shell gsha256sum /usr/local/bin
```

You should of course ensure that `/usr/local/bin` is included in your `$PATH` environment variable.

11.2.2 Creating a research repository

On the server system, you'll want to initialize a bare (i.e., only git files and no working copies of the files) repository accessible and writable by your group.

```
$ git init --bare --shared srj_annex
Initialized empty shared Git repository in /repos/git/srj_annex/
```

If you have `git-annex` set up correctly with your paths, you can then initialize git-annex with a unique name for this repository:

```
$ git annex init origin
init origin ok
(Recording state in git...)
```

Now, clone your repository on your personal computer:

```
$ git clone ssh://server/repos/git/srj_annex
Cloning into 'srj_annex'...
```

That's it: you've now got git-annex-compatible repositories on the server and your computer.

You can also `git annex init` on existing repositories, and add files to those, but because they might already have obnoxiously large files in them it may be better to simply create a new repository. If you're a power user and want to preserve your history but annex old files, you may be able to adapt use this advanced git-filter technique.

11.2.3 Adding files

In the Nemesis environment, we provide a tool *auto-annex* that assists in determining whether files should be added to the repository or added to the `git-annex` store. Instead of calling:

```
$ git add .
```

after creating new files, you should call:

```
$ auto-annex .
```

(assuming the Nemesis environment tools are in your path.)

11.2.4 Adding files manually

Instead of using the `auto-annex` tool, files can be added manually.

When you have a commit to make, it's better to add large files with git-annex before adding them with git. This is because git has to do extensive processing of the file's entire contents, whereas git-annex has a much shorter task to perform.

```
# (move files in or run your code)
$ git annex add *.h5
(Recording state in git...)
$ git add .
$ git commit -m "Initial commit"
[master (root-commit) ac4ed4c] Initial commit
```

Notice that after the first `git annex` command, your HDF5 files have become symlinks that point to a hidden directory stored only on your computer (at the moment). Pushing to the origin only pushes the symbolic link, not the actual data. (Thus, for very large data files that you probably won't ever need to share, you can limit their duplication.) To actually push the annexed files, you should not only push your master branch...:

```
$ git push -u origin HEAD
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 364 bytes | 0 bytes/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To ssh://server/repos/git/srj_annex
 * [new branch]      HEAD -> master
```

...but also copy the annexed files:

```
$ git annex copy --to origin .
(merging origin/git-annex into git-annex...)
(Recording state in git...)
copy celltally.h5 (checking origin...) (to origin...)
....
ok
(Recording state in git...)
```

Now git-annex will tell you that you've safely stored copies of the large file both here and on your remote server:

```
$ git annex whereis celltally.h5
whereis celltally.h5 (2 copies)
    68214887-...... -- [origin]
    9e9236ec-...... -- s3j@local~/srj_annex [here]
```

If more than one person is syncing with your repository, it might be necessary to run `git annex sync` to see all the remote locations where your file is stored.

11.2.5 Managing files

At this point, since git-annex knows that your file is safely stored elsewhere, you can tell it to "drop" the file, or remove the local copy:

```
$ git annex drop celltally.h5
drop celltally.h5 (checking origin...) ok
(Recording state in git...)
```

And now of course, you can no longer access the file on your computer because it's been deleted:

```
$ h5dump celltally.h5
h5dump error: unable to open file "celltally.h5"
```

To get it back by pulling it from the server, you use `git annex get`:

```
$ git annex get celltally.h5
get celltally.h5 (from origin...)
SHA256E-.....h5
      714736 100%   40.72kB/s    0:00:17 (xfer#1, to-check=0/1)
...
ok
(Recording state in git...)
```

By the way, all these transfers are rsync over ssh, so it's efficient, secure, and fast.

If you need to modify an annexed file, which is read-only by design, you'll have to "unlock" it:

```
git annex edit celltally.h5
unlock celltally.h5 (copying...) ok
```

Then you can overwrite it, and the next time you commit, git annex will re-annex the modified file (under a new unique file name):

```
$ git commit -m "Modified cell tally file"
add celltally.h5 ok
ok
(Recording state in git...)
[master 2fb8463] Modified cell tally file
```

Thus both the original file and the modified copy are preserved.

Finally, if you have multiple computers referencing your git annex-ed files, you can run `git annex sync` to ensure that your computer's knowledge of what files are safely annexed is in sync with the server's knowledge.

11.2.6 Cautions

- I've had weird behavior when rebasing with an annexed repository. I'd recommend not rebasing, and just merging your remote and local branches.
- Because annexed files are stored in `.git/objects` rather than the git repostiory itself, it is *imperative* that you call `git annex copy` in addition to `git push` if you plan on deleting and re-pulling the working copy of your repository.

11.3 iPython notebook

iPython notebook is a Python-base interactive analysis environment that we frequently use. We have example files and analysis notebooks in various directories, including `examples/`, `packages/Physica/sce/nb`, and `packages/Omnibus/frontend/nb`.

iPython integrates into Exnihilo via both the Python wrappers (which expose compiled C++ code) and the Omnibus postprocessing suite. The Python wrappers enable users to explore SCALE multigroup and CE data, to build multigroup cross sections, and even to drive SCALE sequences to generate multigroup data.

11.3.1 General conventions

We typically begin each notebook with some boilerplate code that enables various useful packages and default settings:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from omnibus.parser.validator import (MTValidator, NuclideValidator,
                                      to_nuclide)
name_to_mt = MTValidator.to_mt

from srjutils.matplotlib import (article_style, screen_style, poster_style,
                                 grid)
%matplotlib inline
screen_style()

pd.set_option('display.float_format', '{:.3e}'.format)
pd.set_option('display.max_rows', 25)
```

To save a figure for inclusion in a technical article, you should modify the plot settings to use sans-serif fonts and so forth using the `article_style` function:

```
article_style()
plt.plot(x, y)
plt.savefig("everythingisawesome.pdf")
```

11.3.2 Configuration

To create and edit the default iPython notebook:

```
$ ipython profile create
```

If you have a Mac with a "Retina" display or some other high-DPI monitor, you can set iPython to default to producing 2x-resolution images for inline display:

```
$ open ~/.ipython/profile_default/ipython_notebook_config.py
```

and set

```
c.InlineBackend.figure_format = 'retina'
```

Part III

User Guide: Omnibus

Omnibus is the general-purpose front end to Exnihilo. It provides an easy and “traditional” ASCII interface to the deterministic and Monte Carlo solvers in Exnihilo.

CHAPTER

TWELVE

FRONT END INTERFACE

Your Exnihilo installation contains not only an `omnibus` executable (which runs Denovo and Shift) but also additional Python scripts that preprocess user input and program output.

12.1 Running Omnibus

The `omnibus-run` script is able to create an internal Omnibus input file, drive and monitor the `omnibus` executable as it's being run, and postprocess the output.

Note: Unlike most code drivers, omnibus-run is meant to be executed **on the head node** of a cluster rather than on a compute node. Using a machinefile (if the `[RUN=mpi]` option is being used) or pbs submission (for `[RUN=pbs]`), it is able to submit the job to other nodes and monitor the application process.

Tip: For systems such as Titan that have special filespaces (Lustre) that from which the code is executed, the easiest way to ensure that all Omnibus I/O remains on that system is to leave the `.omn` input file on Lustre and call `omnibus-run` from that directory.

12.1.1 Example on a local machine

Suppose you have an input file `batman.omn` on your local machine:

```
[PROBLEM]
name Batman
description "I'm the Batman."

! -- snip -- !

[RUN=mpi]
np 4
```

If Omnibus is installed with MPI, all you need to do is to open a terminal and call:

```
$ omnibus-run batman.omn
```

This runs the following sequence:

1. The preprocessor will validate your input.
2. After successful validation, the precprocessor will write an `xml` intermediate file read in by the `omnibus` binary executable. It will also, if necessary, generate other input files (such as the run tape file used for running on MCNP geometry).

3. The script will launch a local process with arguments like `mpirun -np 4 omnibus batman.xml`, then save the output to the current directory and echo it to the screen.
4. When the `omnibus` binary is complete, it will write out tally data and other program output to several different files.
5. A Python postprocessor reads the output and converts it to a human-readable format, leaving the original output files for later postprocessing.

12.1.2 Example on a cluster using PBS/Torque

In this example, we copy the local machine problem and change the run block (see *[RUN=pbs]*):

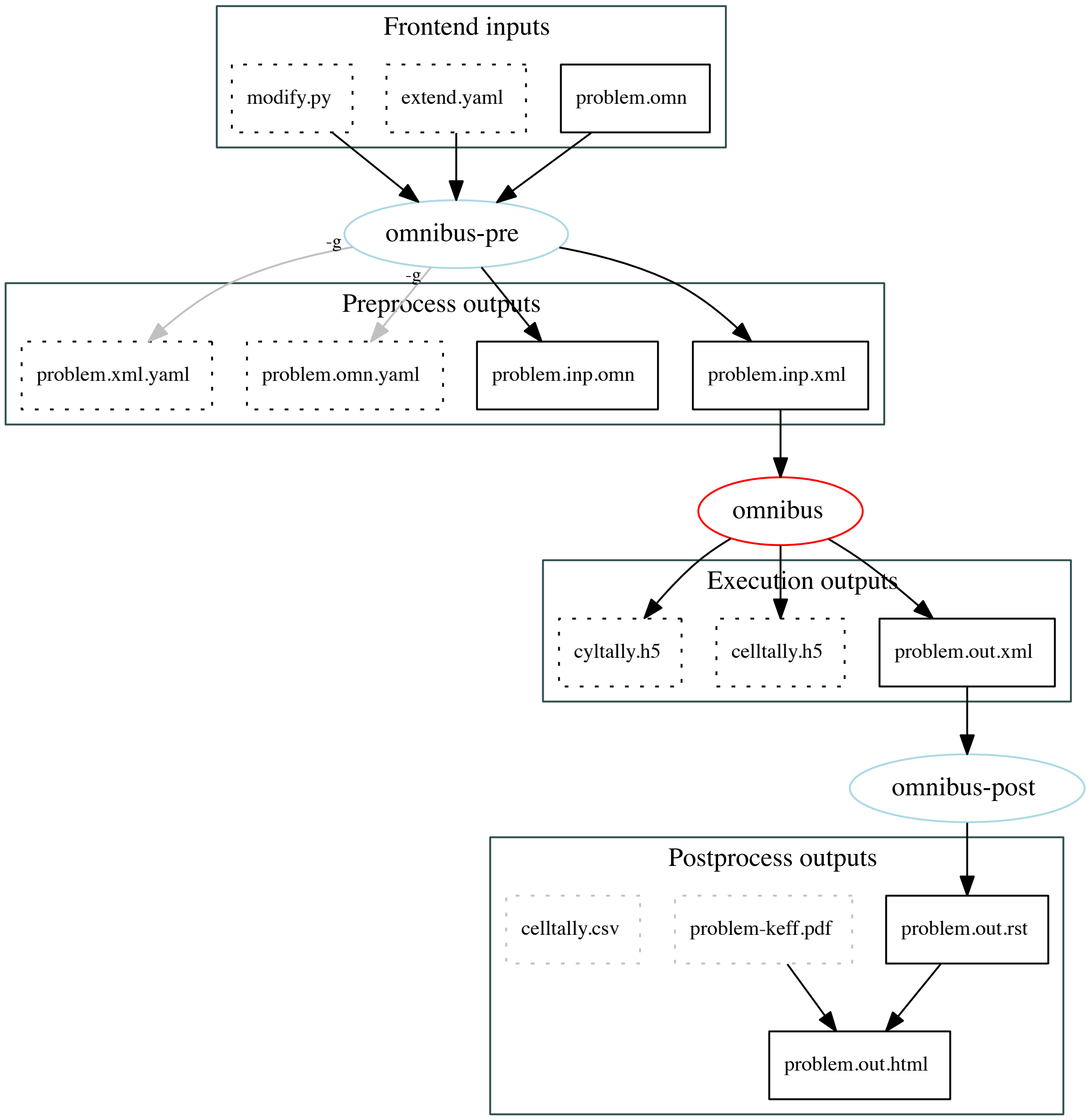
```
[PROBLEM]
name Batman
description "I'm the Batman."

! -- snip -- !

[RUN=pbs]
nodes    1
ppn      32
walltime "1:00:00"
```

When `omnibus-run` is executed on the head node, it will launch `qsub`, monitor the job ID until it begins running on the compute node, and echos output to the screen over an ssh connection. The process on the head node remains almost entirely idle during the problem run, so the user need not worry about incurring the wrath of the system's administrator.

12.2 Omnibus input and output files

Omnibus accepts multiple input formats, writes (possibly multiple) intermediate files, and processes the output into more useful formats.

The following image describes how files are generated and used:



Here, the small black boxes are the typical input/output files, blue circles are scripts, the red circle is the Omnibus executable, and dotted lines are optional files (e.g. having multiple input files or using the `-g` option when calling `omnibus-run` or `omnibus-pre`).

12.2.1 Input files

Omnibus ASCII input files (with the '.omn' extension) are described in *ASCII input*:

```
[PROBLEM]
name "CE pin cell lava_scempp kcode problem"
mode kcode
xml_output kcode.xml
```

```
[GEOMETRY=mcnp]
mcnp mcnp_godiva.mcnp
```

We also support YAML and JSON heirarchical databases, which may appeal more to power users:

```
{
  "problem": {
    "name": "CE pin cell lava_scempp kcode problem",
    "mode": "kcode",
    "xml_output": "kcode.xml"
    },
  "geometry": {
    "_type": "mcnp",
    "input": "mcnp_godiva.mcnp",
  }
}
```

Additionally supported are Python files that can modify an existing (e.g. ASCII-created) database. This last method is extremely powerful for automating repetitious tallies, as demonstrated in this example that creates five similar cylindrical mesh tallies that share an energy grid:

```
import numpy as np

new_tallies = []

neutron_bins = [2e7, 1e5, 1e3, 10, 1, 1e-5]
photon_bins = np.linspace(0, 1e6, 11)[::-1] # 10 linear bins to 1 MeV
reactions   = ["flux"]

targets = [
        # area,       loc,         x,          y,         r
        ( 'PTP', 'FT-A1', -4.66117,  -2.69113, 0.929640,),
        ('SVXF', 'VXF-1',  3.07648,  39.09038, 2.011680,),
        ('SVXF', 'VXF-2', -3.45642,  43.91796, 2.011680,),
        ('SVXF', 'VXF-3', -9.15368,  38.12784, 2.011680,),
        ( 'PTP', 'FT-A1', -4.66117,  -2.69113, 0.929640,),
        ]

# Add each tally to the list
for (area, loc, x, y, r) in targets:
    tal = {
            'name': "%s:%s" % (area, loc),
            'description': "flux in %s target location %s" % (area, loc),
            'reactions': reactions,
            'r': [0.0, r],
            'theta': [0.0, 1.0], # divided by 2pi
            'translate': [x, y, 0],
            'z': [-25.4, 25.4],
            'neutron_bins': neutron_bins,
            'photon_bins': photon_bins,
            }
    new_tallies.append(tal)

# Set all cylindrical tallies
assert 'tally' in db
assert 'cylmesh' not in db['tally']
db['tally']['cylmesh'] = new_tallies
```

12.2.2 Preprocessing output files

The preprocessing step will typically create several files for an input *problem*.omn:

***problem*.json** If using the omnibus-run front end, this will be created: it is a fully processed and reformatted version of the problem input.

***problem*.inp.omn** If using the omnibus-run front end, this will be created: it is a fully processed and reformatted version of the problem input.

***problem*.inp.xml** The Teuchos ParameterList XML file is read by the omnibus executable.

***problem*.inp.xml.json** This file is simply a more-readable version of the XML file, produced when the -g option is enabled for debugging.

Additionally, if MCNP geometry, physics, or sources are being used, *run tape* files will be generated. Finally, if a [RUN=pbs] block is present, a *problem*.pbs submission script will be generated.

12.2.3 Executable output files

All output file names are generated from user input (although the XML output name defaults to *problem*.out.xml). The post-processed *problem*.html file will contain a list of the created files and their descriptions.

12.3 Running Omnibus manually

The relationship between the various input files and exectuables is necessarily complicated, which is why the omnibus-run command is preferred. To generate the Omnibus XML file from an ASCII input, call:

```
$ omnibus-pre my_problem.omn
```

This will create a Teuchos ParameterList XML input file *unikitty*.xml. This parameter list is then run with the Omnibus driver:

```
mpirun -np 16 omnibus my_problem.inp.xml
```

Postprocessing (including plotting keff and Shannon entropy convergence, as well as rendering the XML output into a more human-readable format) is done with the command:

```
omnibus-post my_problem.out.xml
```

12.4 Command line tools

12.4.1 omnibus-run

Run the Omnibus preprocessor, run Omnibus, and run the postprocessor.

Run Omnibus from start to finish.

```
usage: omnibus-run [-h] [--version] [-g] [-c] [-v] [-q] [--very-quiet]
                   [--silent]
                   [--log {None,DEBUG,STATUS,INFO,WARNING,ERROR,CRITICAL}]
                   [--doc]
                   [inp [inp ...]]
```

Positional arguments:

inp	Input file names (omnibus, yaml, and/or python).

Options:

--version	show program's version number and exit
-g=False, --debug=False	Enable extended debug assertions
-c=False, --clobber=False	Overwrite exiting output files rather than renaming them.
-v=STATUS, --verbose=STATUS	Print all debug messages
-q, --quiet	Only print informational and warning messages
--very-quiet	Only print warning messages
--silent	Print messages only on failure
--log	Create a log file with the given verbosity Possible choices: None, DEBUG, STATUS, INFO, WARNING, ERROR, CRITICAL
--doc=False	Print documentation and exit.

Exnihilo version (UNKNOWN)

12.4.2 omnibus-pre

Generate an XML input file for Omnibus, validating input along the way.

Preprocess Omnibus input files.

```
usage: omnibus-pre [-h] [--version] [-g] [-c] [-v] [-q] [--very-quiet]
                   [--silent]
                   [--log {None,DEBUG,STATUS,INFO,WARNING,ERROR,CRITICAL}]
                   [-o OUTPUT] [--doc]
                   [inp [inp ...]]
```

Positional arguments:

inp	Input file names (omnibus, json, yaml, and/or python).

Options:

--version	show program's version number and exit
-g=False, --debug=False	Enable extended debug assertions
-c=False, --clobber=False	Overwrite exiting output files rather than renaming them.
-v=STATUS, --verbose=STATUS	Print all debug messages
-q, --quiet	Only print informational and warning messages
--very-quiet	Only print warning messages
--silent	Print messages only on failure
--log	Create a log file with the given verbosity Possible choices: None, DEBUG, STATUS, INFO, WARNING, ERROR, CRITICAL
-o, --output	Output filename (xml, json, omn, yaml)

--doc=False	Print documentation and exit.

Exnihilo version (UNKNOWN)

12.4.3 omnibus

The actual Omnibus binary executable.

```
usage: omnibus [--version] xml_input
```

Positional arguments:

`xml_input`	Path to the XML parameter input file.

Options:

`--version`	Show usage information and exit.

12.4.4 omnibus-post

Postprocess tally output.

Post-process Omnibus output.

```
usage: omnibus-post [-h] [--version] [-g] [-c] [-v] [-q] [--very-quiet]
                    [--silent]
                    [--log {None,DEBUG,STATUS,INFO,WARNING,ERROR,CRITICAL}]
                    [--fast]
                    outp
```

Positional arguments:

outp	Omnibus XML output file names

Options:

--version	show program's version number and exit
-g=False, --debug=False	Enable extended debug assertions
-c=False, --clobber=False	Overwrite exiting output files rather than renaming them.
-v=STATUS, --verbose=STATUS	Print all debug messages
-q, --quiet	Only print informational and warning messages
--very-quiet	Only print warning messages
--silent	Print messages only on failure
--log	Create a log file with the given verbosity Possible choices: None, DEBUG, STATUS, INFO, WARNING, ERROR, CRITICAL
--fast=False	Undocumented

Exnihilo version (UNKNOWN)

12.5 Advanced execution through Python

Because the Omnibus front end is simply a Python script, it's possible to drive it through another Python script. For example, the shell command

```
$ omnibus-run inyoka.omn
```

can be written as the following python script:

```
from omnibus.scripts import omnibus_run

omnibus_run.run(["inyoka.omn"])
```

The first argument to `run` takes a list, because multiple files can be passed to the input just like with the scripted front end. An additional and very nifty feature is that a python *function* can also be passed to modify the database, just like a Python script can be passed through the command line:

```
from omnibus.scripts import omnibus_run

def change_name(db):
    db['problem']['name'] = "Something else"

omnibus_run.run(["inyoka.omn", change_name])
```

This opens up exciting possiblities of automating scaling studies and parameter studies. For example, this script performs a weak scaling run:

```
from collections import defaultdict
import json
import os

from omnibus.scripts import omnibus_run
from omnibus.utils import working_dir

# Number of histories
histories_per_core = 1e4

data = defaultdict(list)

# We run in a subdirectory, so save the absolute path to our input
omn_input = os.path.abspath("omn_input.omn")

for num_procs in range(1, 8+1):
    def np_setter(db):
        # Adjust number of cores
        db['run'] = {
                '_type': "mpi",
                'np': num_procs,
                }

        # Adjust number of histories
        db['shift']['np'] = num_procs * histories_per_core

    # Execute in a subdirectory and extract the postprocessed out.xml file
    with working_dir("np-%01d" % (num_procs)):
        manager = omnibus_run.run_or_pp([omn_input, np_setter])

    data['cores'].append(manager.num_procs)
    data['histories'].append(manager.num_histories)
    data['solve_time'].append(manager.get_timing("shift",
                                    "shift::Fixed_Source_Solver.solve"))
# Save timing results
with open('results.json', 'w') as f:
    json.dump(dict(data), f)
```

The special `run_or_pp` command will reuse any existing `.out.xml` file if available, so this script can be resumed if canceled partway. The result is a lovely file of solution times:

```
{"cores": [1, 2, 3, 4, 5, 6, 7, 8],
 "histories":  [10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000],
 "solve_time": [66.6750500202179, 71.05331420898438, 73.68022584915161,
     74.62563920021057, 75.13657093048096, 75.84799098968506,
     76.0614001750946, 76.82996106147766]}
```

which can then be plotted or analyzed:

```
import json
import pandas as pd
with open("results.json") as f:
    data = json.load(f)

results = pd.DataFrame.from_dict(data)
results.index = results.pop("cores")

results['solve_time'].plot()
```

12.6 Parameter list explanation

The `omnibus` executable runs using a heirarchical XML parameter list. The necessary complexity of the parameter list makes it nearly prohibitive for a user to write it from scratch. However, some knowledge of the underlying XML parameter list structure is useful in understanding the design choices made for the Omnibus frontend.

The parameter list tree built by Omnibus consists of three general types of elements:

- Parameters: simple elements with a name and a particular range of allowed values (e.g. a string, a real number between zero and one, or a list of integers);
- Databases: containers of parameters, sub-databases, and sublists; and
- Sublists: a list of congruent databases (e.g. materials in a problem).

Each database has to conform to a specific "class", such as the PROBLEM database or the SHIFT database. Some databases have different "types": these are viewed as interchangeable by the enclosing database, but the different types may have different parameters associated with them. For example, the "SHAPE" sub-database of the SOURCE database could be a "box" type or a "cylinder" type. The SOURCE database requires a single shape sub-database, but each of those two types has a different set of required parameters (the dimensions of the box versus the width and height of the cylinder).

12.7 Developer notes

Exnihilo developers will need to modify the Omnibus input description to expose new transport capabilities to the front end. The input description is contained in the Python files at `Omnibus/frontend/omnibus/omn`. The root-level database is located in `root.py`, and sub-databases are located alongside it.

To update the ASCII input regression problems when the input database changes, run `make regression_update` from `Exnihilo/packages/Omnibus/frontend` inside the build directory.

12.7.1 Updating documentation

Additional documentation (besides the one-line parameter descriptions) can be added by expanding the files inside `Omnibus/frontend/doc/detail`. All of the `.rst` files will be read and treated equally; the multiple .rst files are just for convenience in editing and organizing the files.

The pseudo-directive `.. OMNIDOC {location}` is used to insert the following documentation (until the next `.. OMNIDOC` directive or the end of the file). The `{location}` is a simple syntax that is parsed to determine where in the Omnibus input documentation the contents should be inserted:

- Use `.` to separate subdatabases.
- Use = to specify a particular subtype for a database.
- The final entry can be a parameter or command, or it can be `_intro` to be printed at the top of a database description, or `_concl` to be added to the bottom.

Example locations are:

```
problem.mode
geometry=rtk._intro
source.shape=box.box
shift.decomposition.overlap
```

To regenerate the Omnibus input documentation, run `make omnidoc` from `Exnihilo/packages/Omnibus/frontend` inside the build directory.

CHAPTER

THIRTEEN

OMNIBUS ASCII INPUT FORMAT

The Omnibus ASCII input is a human-readable, minimal input syntax for Omnibus. The underlying Omnibus data structure is hierarchical, and the ASCII input is designed to flatten the hierarchy. The input consists of "blocks" of input data, each of which represents a database, and cards, which consist of parameters and "commands", which generate parameters or perform other functions.

13.1 Blocks

Block titles have the formats

```
[CLASS]
[CLASS=type]
[CLASS name]
[PARENT][CLASS=type name]
[GRANDPARENT][PARENT][CLASS=type name]
```

which embeds the location in the hierarchy, the database class, the database type, and the name of this particular instance of the database class. The "name" (which requires a value with only letters, numbers, and the underscore) is simply a shorthand for declaring the block and adding a "name" parameter. The class type is required for databases that have multiple allowed types (e.g. geometry and physics) but disallowed for types that do not. The block will be inserted inside the last instance of the [GRANDPARENT][PARENT] block.

Whitespace in block titles, as well as capitalization for the class and type attributes, is ignored.

Note: The exact regular expression used to match titles is:

```
^(?:\[\s*(\w+)(?:\s*=\s*(\w+))?\s*(\w+)?\]\s*)+$
```

13.2 Cards

Cards are started on a new line; an indentation of **four or more spaces** is treated as a continuation of the previous card. Spaces separate values in a parameter list or arguments in a command. For strings, quotation marks can be used to treat whitespace as standard characters. The backslash can be used to escape quotation marks inside a quoted string.

For example, these two parameters demonstrate the correct usage of whitespace:

```
param This is a list of seven parameters.
param "This is a single parameter with an \" embedded quotation mark."
```

Tip: One common input error is to mistake a small indentation on the next line for a continuation. This statement declares three parameters inside a block:

```
[WAYNE]
something value value
  business business numbers
  is this working
```

whereas this is one parameter with multiple values:

```
[WAYNE]
something value value
    business business numbers
    is this working
```

Using the syntax highlighting files for Vim and Emacs provided in the `Exnihilo/environment` directory (see *Development Environment*) will make such errors very obvious.

13.3 Other features

Whitespace (outside of the initial line indentation and when within quotation marks) is generally flexible and ignored.

Inside numerical lists, standard MCNP interpolation/repetition shorthands "I", "ILOG", "M", and "R" are implemented:

```
x_coordinates 1 2I 4
```

is interpolated to form:

```
x_coordinates 1 2 3 4
```

Tip: The `vacuum_omnibus_input` script will read your Omnibus input file, reformat it, and rewrite it. If the input and output are not logically the same, there is a subtle syntax error in the input file (e.g. not indenting when continuing). This tool only parses the input file; it does no expansion, validation, or defaulting.

Tip: To view a validated and reformatted ASCII version of your input, you can explicitly tell the preprocessor to save an `.omn` file:

```
$ omnibus-pre problem.omn -o problem.validated.omn
```

CHAPTER

FOURTEEN

OMNIBUS INPUT DATABASE SPECIFICATION

version 5.3 (branch ‘doc’ #60625037 on 2015FEB17)

date 2015-02-17 22:08:00

The Omnibus input is split into a heirarchy of blocks. Each of the first-level blocks (and the overall problem input file) are described in the following section:

14.1 Omnibus input file

14.1.1 Sub-databases

Name and type	Frequency
problem	Exactly once
response = { *histogram*, *interpolated*}	Zero or more
tally	Optional
source = { *separable*, *fissionmesh*, *mesh*}	At least one
geometry = { *mcnp*, *scale*, *rtk*, *mesh*}	Exactly once
comp	Optional
physics = { *mg*, *sce*, *smg*}	At least one
depletion	Optional
shift	Optional
denovo	Optional
manualww	Optional
run = { *serial*, *mpi*, *pbs*}	Optional

14.1.2 Additional defaults

- If running a kcode problem with only Denovo, no source needs to be defined.

14.1.3 Additional restrictions

- The ‘denovo’ and ‘manualww’ databases are mutually exclusive..
- No transport will be run if [SHIFT] and [DENOVO] databases are both missing..

14.2 [PROBLEM]

The problem database specifies top-level information about the problem being run. It includes the output file name, a unique problem identifier, and overall solution technique.

14.2.1 Parameters

`name`
Descriptive problem name.

Default ‘Untitled’

Type string

`mode`
Problem mode.

Valid modes are:

kcode Solve the *k*-eigenvalue problem for criticality safety or reactor physics analysis.

forward Solve a fixed-source problem for shielding calculations etc.

check Do not transport any particles: parse the input, generate the necessary files, set up the problem geometry and physics, then exit.

Type ‘kcode’, or ‘forward’

14.2.2 Advanced parameters

These parameters are not meant for typical use.

`xml_output / xml`
Destination path for the XML output file.

Type file path to write (extension ‘.xml’)

`pid`
Unique identifier automatically set for this problem run.

The problem identifier (pid) is a unique string generated by the Omnibus preprocessor to ensure that input and output files are properly correlated. The problem ID value added to the XML input file is copied to the XML output file as well as all relevant HDF5 output files. It’s comprised of the problem execution date and a randomly generated unique identifier string (UUID).

Type string

14.2.3 Additional defaults

- Default the XML output path to *`input`*`.out.xml`.
- Set the unique problem identifier for this run.

14.3 [RESPONSE]

The RESPONSE block is for defining energy- and particle-dependent responses such as dose conversion factors. Each response can be used multiple times in the tallies.

14.4 [RESPONSE=histogram]

14.4.1 Parameters

`name`
Short title or label for response.

Type string

`description`
Optional longer descriptive string.

Default
‘’

Type string

`energy_bounds / e`
Lowest bound plus upper bounds for the histograms.

Type list of monotonically increasing floats

Units eV

`response_values / vals`
Histogram response values.

Type list of floats

`particle_type / pt`
Particle type to apply this response.

Type particle type (‘p’,’n’,’np’,’neutron’,’photon’)

14.4.2 Additional restrictions

- Size of energy bounds must be one greater than the number of response values.

14.4.3 Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
energy_bounds		energy_points

14.5 [RESPONSE=interpolated]

14.5.1 Parameters

name
Short title or label for response.

Type string

description
Optional longer descriptive string.

Default
‘’

Type string

interpolation_type / interp
Type of interpolated response.

Type ‘linear’, ‘log_lin’, ‘lin_log’, or ‘log_log’

energy_points / e
Energy points.

Type list of monotonically increasing floats

Units eV

response_values / vals
Response values at energy points.

Type list of floats

particle_type / pt
Particle type to apply this response.

Type particle type (‘p’,’n’,’np’,’neutron’,’photon’)

14.5.2 Additional restrictions

- Response energies must be list of positive floatsfor `log_log` and `log_lin` response types.
- Response energies must be list of positive floatsfor `log_log` and `lin_log` response types.

14.6 [TALLY]

The tally database specifies tallies and problem diagnostics.

General tallies support a common set of attributes, including a name and optional description, a list of reactions to tally, a list of *[RESPONSE]* objects, and neutron and photon energy bin boundaries.

Note: Unlike other Monte Carlo transport codes, the bin boundaries in Omnibus are truly boundaries, not upper or lower energies. Thus the number of energy bins will be one less than the number of bounds. To reproduce the behavior of MCNP and Monaco, which tally from the cutoff energy to the lowest given energy, you must explicitly add the cutoff energy as the lowest energy bound.

Because multiple tallies can write to the same files, file names and prefixes are saved in the main *[TALLY]* block.

14.6.1 Sub-databases

Name and type	Frequency
mesh	Zero or more
cylmesh	Zero or more
cell	Zero or more
diagnostic = { *pathlength*, *collision*, *source*, *history*, *debug* }	Zero or more
sensitivity	Optional

14.6.2 Parameters

`mesh_silo_output`
Prefix name for mesh tally silo output files.

Default 'meshtally'

Type string

Applicability when at least one MESH tally is present.

`cell_hdf5_output / cell_out`
Name of cell tally hdf5 output file.

Default 'celltally.h5'

Type file path to write (extension '.h5') (empty value allowed)

Applicability when at least one CELL tally is present.

`cylmesh_hdf5_output / cyl_out`
Name of small cylindrical tally hdf5 output file.

Default 'cyltally.h5'

Type file path to write (extension '.h5') (empty value allowed)

Applicability when at least one CYLMESH tally is present.

`mesh_hdf5_output / mesh_out`
Name of mesh tally hdf5 output file.

Default 'meshtally.h5'

Type file path to write (extension '.h5') (empty value allowed)

Applicability when at least one MESH tally is present.

14.6.3 Additional restrictions

- [TALLY] block must contain at least one sublist (MESH, CELL, etc.).

14.6.4 [TALLY][MESH]

Shift supports multiple structured Cartesian path length mesh tallies. The mesh can be defined over all or part of the physical problem geometry. The Cartesian mesh tally is output at the end of a problem in Silo format for easy analysis using VisIt, and in HDF5 format for analysis.

Parameters

`name`
Short title or label for the tally.

Type string

`description / desc`
Optional longer descriptive string.

Default
‘’

Type string

`reactions / rxn`
Reactions to calculate for this tally.

Default [’flux’]

Type list of ‘flux’, ‘total’, ‘absorption’, ‘scattering’, ‘fission’, ‘nu_fission’, ‘kappa_sigma’, or ‘kerma’

`responses / resp`
Responses for this tally.

Default []

Type list of strings

`normalization`
Constant multiplicative factor to apply to tally results.

Default 1.0

Type positive real number

`neutron_bins / nbins`
Energy bin boundaries for neutrons.

Default []

Type nonnegative floats in decreasing order

Units eV

`photon_bins / pbins`
Energy bin boundaries for photons.

Default []

Type nonnegative floats in decreasing order

Units eV

x
Mesh tally coordinates along the X axis.

Type list of two or more monotonically increasing floats

Units cm

y
Mesh tally coordinates along the Y axis.

Type list of two or more monotonically increasing floats

Units cm

z
Mesh tally coordinates along the Z axis.

Type list of two or more monotonically increasing floats

Units cm

Additional restrictions

- Response names are validated against [RESPONSE] blocks.

14.6.5 [TALLY][CYLMESH]

Particles can be tracked on a translated, rotated cylinder broken into (r, z, θ) mesh cells.

Parameters

name
Short title or label for the tally.

Type string

description / desc
Optional longer descriptive string.

Default
‘’

Type string

reactions / rxn
Reactions to calculate for this tally.

Default [’flux’]

Type list of ‘flux’, ‘total’, ‘absorption’, ‘scattering’, ‘fission’, ‘nu_fission’, ‘kappa_sigma’, or ‘kerma’

responses / resp
Responses for this tally.

Default []

Type list of strings

`normalization`
Constant multiplicative factor to apply to tally results.

Default 1.0

Type positive real number

`neutron_bins / nbins`
Energy bin boundaries for neutrons.

Default []

Type nonnegative floats in decreasing order

Units eV

`photon_bins / pbins`
Energy bin boundaries for photons.

Default []

Type nonnegative floats in decreasing order

Units eV

`r`
Radial mesh coordinates.

Type list of two or more monotonically increasing floats

Units cm

`theta`
Theta mesh coordinates.

Default [0.0, 1.0]

Type list of two or more monotonically increasing floats

Units revolutions

`z`
Mesh coordinates along the Z axis.

Type list of two or more monotonically increasing floats

Units cm

`rotate / rot`
Rotation matrix.

Default [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0]

Type row-major flattened matrix

`translate / trans`
Translation vector (applied after rotation).

Default [0.0, 0.0, 0.0]

Type xyz coordinates

Additional restrictions

- Response names are validated against [RESPONSE] blocks.

14.6.6 [TALLY][CELL]

Geometry cells and unions of cells are tallied using a hash table, allowing constant-time scaling with respect to the number of total cells being tallied.

The recommended way to tally multiple particle spectra in the same cell is to use a single tally.

```
[TALLY][CELL energybinned]
description "5n3g energy-binned tally."
reactions flux
cells 1 100 4:5:6
neutron_bins  1e7 1e6 1e3 10 1 1e-3
photon_bins   2e7 1e6 1e5 1e3
```

Parameters

name
Short title or label for the tally.

Type string

description / desc
Optional longer descriptive string.

Default
‘’

Type string

reactions / rxn
Reactions to calculate for this tally.

Default [’flux’]

Type list of ‘flux’, ‘total’, ‘absorption’, ‘scattering’, ‘fission’, ‘nu_fission’, ‘kappa_sigma’, or ‘kerma’

responses / resp
Responses for this tally.

Default []

Type list of strings

normalization
Constant multiplicative factor to apply to tally results.

Default 1.0

Type positive real number

neutron_bins / nbins
Energy bin boundaries for neutrons.

Default []

Type nonnegative floats in decreasing order

Units eV

photon_bins / pbins
Energy bin boundaries for photons.

Default []

Type nonnegative floats in decreasing order

Units eV

Advanced parameters

These parameters are not meant for typical use.

`union_cells`
Flattened list of cells in each union.

Type list of integers

`union_lengths`
Number of cells per union in the above list.

Type list of integers

Commands

These commands generate one or more parameters.

`cells`
Generate 'union_cells' and 'union_lengths' from colon-separated unions.

Additional restrictions

- Response names are validated against [RESPONSE] blocks.

14.6.7 [TALLY][DIAGNOSTIC=pathlength]

The path length diagnostic produces a distribution of the path length traveled by a particle between events. Note that this is not the same as the true path length distribution (distance between collisions), as the events considered by Shift include material boundary crossings, problem boundary crossings, etc.

Parameters

`pl_bins`
Lower bin boundaries for the traversed path lengths in each event.

Type list of positive floats

`event_bins`
Lower bin boundaries for the number of events per history.

Type list of monotonically increasing nonnegative integers

14.6.8 [TALLY][DIAGNOSTIC=collision]

The collision diagnostic tallies the number of collisions per history as a function of material, nuclide, and reaction ID.

Parameters

`output`
Path to collision diagnostic hdf5 output file.

Default 'collisions.h5'

Type file path to write (extension '.h5') (empty value allowed)

Additional restrictions

- This tally is only compatible with SCE physics.

14.6.9 [TALLY][DIAGNOSTIC=source]

This diagnostic tally is provided to calculate the source density binned into spatial cells. It also calculates the *particle* source density (i.e., binning $n(\vec{r})$ rather than $wn(\vec{r})$) to assist in the construction of biased sources.

Parameters

`x`
Mesh tally coordinates along the X axis.

Type list of two or more monotonically increasing floats

Units cm

`y`
Mesh tally coordinates along the Y axis.

Type list of two or more monotonically increasing floats

Units cm

`z`
Mesh tally coordinates along the Z axis.

Type list of two or more monotonically increasing floats

Units cm

`silo_output`
Path to Silo output file (no extension).

Default 'source'

Type string

14.6.10 [TALLY][DIAGNOSTIC=history]

The history diagnostic tallies every event in a particle's lifetime. Currently, all particle histories are tallied, and only one processor writes the histories.

Parameters

`history_hdf5_output / output`
Destination file for history events.

Default 'history.h5'

Type file path to write (extension '.h5') (empty value allowed)

`node`
Node from which to save history events.

Default 0

Type non-negative integer

14.6.11 [TALLY][DIAGNOSTIC=debug]

The debug diagnostic currently records the number of events per particle history, but it will be extended to provide other useful high-level debug information. The output data is written to the omnibus output XML file and converted in the postprocessor.

14.6.12 [TALLY][SENSITIVITY]

The sensitivity tally allows sensitivities to cross sections and other input to be calculated using advanced methods.

Note: Currently sensitivity tallies can only be run in serial mode.

Parameters

`neutron_bins / nbins`
Energy bin boundaries for neutrons.

Default []

Type nonnegative floats in decreasing order

Units eV

`method / cet`
Sensitivity coefficient calculation mode.

Default 'ce_tsunami'

Type 'ce_tsunami', 'clutch', or 'ifp'

`latent_generations / cfp`
Number of latent generations for IFP/F*(r) calculation.

Default 1

Type positive integer

Applicability when S/U method uses an F* mesh.

Advanced parameters

These parameters are not meant for typical use.

`sensitivity_output`
Destination path for the sensitivity tally output file (.sdf).

Type file path to write (extension '.sdf')

Additional defaults

- Default the SDF output path to INPUT.sdf.

Additional restrictions

- This tally is only compatible with SCE physics.

14.7 [SOURCE]

The source database specifies the source particle distribution for a fixed-source problem. Currently, only separable, uniform sources are supported. All source strengths are normalized to unity. To adjust the effective strength of a source (e.g. to set a multiplicative counts-per-second factor), modify the `normalization` property of the tallies.

14.8 [SOURCE=separable]

14.8.1 Sub-databases

Name and type	Frequency
shape = { *box*, *cylinder*, *cylindershell*, *sphere*, *sphereshell*, *point*}	Exactly once
energy = { *histogram*, *mono*, *lines*, *watt*}	Exactly once
angle = { *isotropic*, *mono*}	Exactly once

14.8.2 Parameters

`name`
Short title or label for the source.

Default 'source'

Type string

`description / desc`
Optional longer descriptive string.

Default
''

Type string

`strength / q`
Source strength.

Default 1.0

Type positive real number

Units particles/s

`particle_type / pt`
Particle type to emit.

Type particle type ('p','n','np','neutron','photon')

`fissionable_only / fis`
Whether particles are only emitted in fissionable regions.

Type boolean

14.8.3 Additional defaults

- When running an eigenvalue problem, Default to an isotropic distribution and a Watt (U-235) energy spectrum, and only emit particles inside of fissionable cells.

14.8.4 [SOURCE=separable][SHAPE=box]

Parameters

`xmin`
Minimum x-coordinate of box source.

Type real number

`xmax`
Maximum x-coordinate of box source.

Type real number

`ymin`
Minimum y-coordinate of box source.

Type real number

`ymax`
Maximum y-coordinate of box source.

Type real number

`zmin`
Minimum z-coordinate of box source.

Type real number

`zmax`
Maximum z-coordinate of box source.

Type real number

Commands

These commands generate one or more parameters.

box
Expand into parameters `xmin`, `xmax`, `ymin`, `ymax`, `zmin`, `zmax`.

14.8.5 [SOURCE=separable][SHAPE=cylinder]

Parameters

axis
Axis along regular cylinder source.

Type axis ('x','y','z')

origin_x / xo
X-coordinate of regular cylinder source origin.

Type real number

origin_y / yo
Y-coordinate of regular cylinder source origin.

Type real number

origin_z / zo
Z-coordinate of regular cylinder source origin.

Type real number

radius / r
Radius of regular cylinder source.

Type real number

height / h
Height of regular cylinder source.

Type real number

Commands

These commands generate one or more parameters.

origin
Expand into parameters `origin_x`, `origin_y`, `origin_z`.

14.8.6 [SOURCE=separable][SHAPE=cylindershell]

Parameters

axis
Axis along regular cylinder shell source.

Type axis ('x','y','z')

origin_x / xo
X-coordinate of regular cylinder shell source origin.

Type real number

`origin_y / yo`
Y-coordinate of regular cylinder shell source origin.

Type real number

`origin_z / zo`
Z-coordinate of regular cylinder shell source origin.

Type real number

`inner_radius / ir`
Inner radius of regular cylinder shell source.

Type real number

`outer_radius / or`
Outer radius of regular cylinder shell source.

Type real number

`height / h`
Height of regular cylinder shell source.

Type real number

Commands

These commands generate one or more parameters.

`origin`
Expand into parameters `origin_x`, `origin_y`, `origin_z`.

14.8.7 [SOURCE=separable][SHAPE=sphere]

Parameters

`origin_x / xo`
X-coordinate of sphere source origin.

Type real number

`origin_y / yo`
Y-coordinate of sphere source origin.

Type real number

`origin_z / zo`
Z-coordinate of sphere source origin.

Type real number

`radius / r`
Radius of sphere source.

Type real number

Commands

These commands generate one or more parameters.

`origin`
Expand into parameters `origin_x`, `origin_y`, `origin_z`.

14.8.8 [SOURCE=separable][SHAPE=sphereshell]

Parameters

`origin_x / xo`
X-coordinate of sphere shell source origin.

Type real number

`origin_y / yo`
Y-coordinate of sphere shell source origin.

Type real number

`origin_z / zo`
Z-coordinate of sphere shell source origin.

Type real number

`inner_radius / ir`
Inner radius of sphere shell source.

Type real number

`outer_radius / or`
Outer radius of sphere shell source.

Type real number

Commands

These commands generate one or more parameters.

`origin`
Expand into parameters `origin_x`, `origin_y`, `origin_z`.

14.8.9 [SOURCE=separable][SHAPE=point]

Parameters

`x`
X position of point source.

Type real number

`y`
Y position of point source.

Type real number

`z`
Z position of point source.

Type real number

Commands

These commands generate one or more parameters.

point
Expand into parameters `x`, `y`, `z`.

14.8.10 [SOURCE=separable][ENERGY=histogram]

Parameters

energy / e
Histogram lower energy bin bounds.

Type list of non-negative monotonically increasing floats

Units eV

strength / q
Average source strength inside each bin.

Type list of non-negative floats

Additional restrictions

- Histogram source must have one more `energy` value than `strength`.

Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
energy		domain
strength		range

14.8.11 [SOURCE=separable][ENERGY=mono]

Parameters

energy / e
Source energy.

Type positive real number

14.8.12 [SOURCE=separable][ENERGY=lines]

Parameters

energy / e
Individual line energies.

Type list of positive floats

Units eV

`strength / q`
Strength corresponding to each source.

Type list of non-negative floats

Additional restrictions

- Line source must have the same number of `energy` and `strength`.

14.8.13 [SOURCE=separable][ENERGY=watt]

Parameters

`a`
Value for the 'a' constant in Watt equation.

Type positive real number

Units MeV

`b`
Value for the 'b' constant in Watt equation.

Type positive real number

Units 1/MeV

Additional defaults

- Watt spectrum defaults to a=0.965 and b=2.29 (U-235 fission).

14.8.14 [SOURCE=separable][ANGLE=isotropic]

14.8.15 [SOURCE=separable][ANGLE=mono]

Parameters

`direction / dir`
Direction source particles are emitted.

Type xyz coordinates

14.9 [SOURCE=fissionmesh]

14.9.1 Sub-databases

Name and type	Frequency
energy = { *histogram*, *mono*, *lines*, *watt*}	Exactly once

14.9.2 Parameters

`name`
Short title or label for the source.

Default 'source'

Type string

`description / desc`
Optional longer descriptive string.

Default

''

Type string

`strength / q`
Source strength.

Default 1.0

Type positive real number

Units particles/s

`particle_type / pt`
Particle type to emit.

Type particle type ('p','n','np','neutron','photon')

`fissionable_only / fis`
Whether particles are only emitted in fissionable regions.

Type boolean

`mesh_tally_input / input`
Name of file containing fission source distribution.

Type file path for reading (extension '.h5')

`mesh_tally_name`
Name of fission source tally in file.

Type string

14.9.3 Additional defaults

- Default to an isotropic distribution and a Watt (U-235) energy spectrum, and only emit particles inside of fissionable cells.

14.9.4 [SOURCE=fissionmesh][ENERGY=histogram]

Parameters

`energy / e`
Histogram lower energy bin bounds.

Type list of non-negative monotonically increasing floats

Units eV

strength / q
Average source strength inside each bin.

Type list of non-negative floats

Additional restrictions

- Histogram source must have one more `energy` value than `strength`.

Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
energy		domain
strength		range

14.9.5 [SOURCE=fissionmesh][ENERGY=mono]

Parameters

energy / e
Source energy.

Type positive real number

14.9.6 [SOURCE=fissionmesh][ENERGY=lines]

Parameters

energy / e
Individual line energies.

Type list of positive floats

Units eV

strength / q
Strength corresponding to each source.

Type list of non-negative floats

Additional restrictions

- Line source must have the same number of `energy` and `strength`.

14.9.7 [SOURCE=fissionmesh][ENERGY=watt]

Parameters

a
Value for the ‘a’ constant in Watt equation.

Type positive real number

Units MeV

b

Value for the ‘b’ constant in Watt equation.

Type positive real number

Units 1/MeV

Additional defaults

- Watt spectrum defaults to a=0.965 and b=2.29 (U-235 fission).

14.10 [SOURCE=mesh]

14.10.1 Parameters

name

Short title or label for the source.

Default ‘source’

Type string

description / desc

Optional longer descriptive string.

Default

‘’

Type string

strength / q

Source strength.

Default 1.0

Type positive real number

Units particles/s

particle_type / pt

Particle type to emit.

Type particle type (‘p’,’n’,’np’,’neutron’,’photon’)

fissionable_only / fis

Whether particles are only emitted in fissionable regions.

Type boolean

mesh_source_input / input

Name of file containing mesh source distribution.

Type file path for reading (extension ‘.h5’)

14.11 [GEOMETRY]

Shift provides adapters for several geometry types.

Note: For most geometry types, Shift allows cell volumes to be manually input with the “volumes” and “volume_cells” keywords:

```
volumes       1.0 2.34  0.5
volume_cells  1   200   15
```

Here the cells are the “cell labels” (e.g. the cell card IDs in MCNP) and the volumes are the corresponding volumes in cm^3.

14.12 [GEOMETRY=mcnp]

MCNP support is provided through the Lava plugin, developed at ORNL by Scott Mosher. Lava is a C interface to MCNP routines. It requires MCNP to generate a run tape from an input file. Omnibus will automatically execute MCNP and generate the run tape.

Note: If using Shift cell tallies with MCNP geometry, the user must include a cell tally or input volumes for desired cells *in the MCNP input* to for volumes to automatically be propagated into Shift.

14.12.1 Parameters

`volumes`
Provide or override volumes for cells in the geometry.

Default []

Type list of positive floats

Units cc

`volume_cells`
Cell labels corresponding to the given volume overrides.

Default []

Type list of integers

14.12.2 Advanced parameters

These parameters are not meant for typical use.

`runtpe_path`
Path to the MCNP runtpe file.

Type file path for reading

14.12.3 Commands

These commands generate one or more parameters.

`input`
Generate an MCNP runtpe file and set `runtpe_path`.

14.13 [GEOMETRY=scale]

The SCALE geometry supports reading directly from a SCALE input file that includes a KENO VI geometry. Contact Rob Lefebvre <lefebvrera@ornl.gov> for the current status of the Atlas geometry package that underlies KENO support. As of December 2014, the following capabilities are not supported:

- Reflecting boundaries
- Hexagonal arrays

14.13.1 Parameters

`input`
Path to the KENO VI input file.

Type file path for reading (extension '.inp')

`volumes`
Provide or override volumes for cells in the geometry.

Default []

Type list of positive floats

Units cc

`volume_cells`
Cell labels corresponding to the given volume overrides.

Default []

Type list of integers

14.14 [GEOMETRY=rtk]

The RTK geometry implemented by Omnibus reads a geometry from an XML input file. In general, the Insilico front-end should be used for creating reactors with the RTK geometry.

14.14.1 Parameters

`input`
Path to the RTK geometry xml file.

Type file path for reading (extension '.xml')

14.15 [GEOMETRY=mesh]

14.15.1 Parameters

`input_type`
Where the mesh geometry is defined.

Default 'hdf5'

Type 'hdf5', or 'manual'

`input`
Path to the mesh geometry hdf5 file.

Type file path for reading (extension '.h5')

Applicability when 'input_type' is 'hdf5'.

`matids`
List of material IDs for each mesh cell.

Type list of nonnegative integers

Applicability when 'input_type' is 'manual'.

`x_planes`
Mesh coordinates along the X axis.

Type list of two or more monotonically increasing floats

Units cm

Applicability when 'input_type' is 'manual'.

`y_planes`
Mesh coordinates along the Y axis.

Type list of two or more monotonically increasing floats

Units cm

Applicability when 'input_type' is 'manual'.

`z_planes`
Mesh coordinates along the Z axis.

Type list of two or more monotonically increasing floats

Units cm

Applicability when 'input_type' is 'manual'.

14.15.2 Additional restrictions

- If more than 1000 mesh cells are present, then HDF5 input must be used.

14.16 [COMP]

The *[COMP]* block allows custom definition of compositions. The matids in each block must correspond to the matids in the problem geometry.

14.16.1 Sub-databases

Name and type	Frequency
material	At least one

14.16.2 [COMP][MATERIAL]

Parameters

`name`
Label for the material.

Type string

`matid`
Internal matid number.

The `matid` is a [0,N)-indexed internal numbering system. The matids used by Shift typically differ from the material names used in the problem physics input (for example, *m10* in an MCNP input deck might correspond to matid=1). Currently, the only way to guarantee this corresponds to a particular material in the geometry input is to use an input that specifies matids explicitly (RTK or mesh geometry). However, by viewing the matid-to-label mapping given in an Omnibus postprocess output for a SCALE or MCNP input geometry, it is possible to figure out for a particular problem what matid corresponds to what material.

Type non-negative integer

`temperature / tmp`
Material temperature.

Type non-negative real number

Units K

`deplete / depl`
Whether the material is depletable.

Default False

Type boolean

`fission / fiss`
Whether the material is fissionable.

Default False

Type boolean

`zaid`
Element IDs (MZZZAAA) in this material.

Type list of positive integers

`nd`
Number densities of each nuclide.

Type list of positive floats

Units atoms/(b-cm)

Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
nd		num_densities
zaid		elements

14.17 [PHYSICS]

Omnibus supports several coupled-physics packages.

14.18 [PHYSICS=mg]

The simple multigroup physics package allows user-input P_0 cross sections for test problems such as C5G7.

14.18.1 Sub-databases

Name and type	Frequency
xs	Zero or more

14.18.2 Parameters

name
Label for the physics.

Default 'mg'

Type string

mg_lib_path
MG library name or path to MG library file.

Default
''

Type string

downscatter
Indicates whether this problem has downscatter only.

Default False

Type boolean

pn_order
Scattering order of problem.

Default 0

Type zero or positive odd integer

comp_hdf5_output / comp_out
Write compositions to the given HDF5 file, or skip if empty.

Default 'compositions.h5'

Type file path to write (extension '.h5') (empty value allowed)

num_groups
Number of energy groups.

Type integer

Applicability when Using manual cross section input.

neutron_bnd
Neutron group boundaries.

Default []

Type positive floats in decreasing order

Applicability when Using manual cross section input.

photon_bnd
Photon group boundaries.

Default []

Type positive floats in decreasing order

Applicability when Using manual cross section input.

14.18.3 Additional restrictions

- XS database specification and 'mg_lib_path' are mutally exclusive.
- Only Pn order = 0 is currently implemented.

14.18.4 Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
downscatter	db	downscatter
mg_lib_path	db	xs_library
neutron_bnd	db	neutron_bnd
num_groups	db	num_groups
photon_bnd	db	photon_bnd
pn_order	db	Pn_order

14.18.5 [PHYSICS=mg][XS]

This subdatabase specifies all cross sections for a single material. If the material is fissionable, all of `fission`, `nu`, and `chi` must be present.

Parameters

name
Label for the material.

Default 'untitled'

Type string

matid
Material ID.

Type non-negative integer

total
Total cross section by group.

Type list of non-negative floats

s0
Isotropic scattering cross section.

Type list of non-negative floats

fission
Fission cross section.

Default []

Type list of non-negative floats

nu
Neutron production.

Default []

Type list of non-negative floats

chi
Fission spectrum.

Default []

Type list of non-negative floats

14.19 [PHYSICS=sce]

SCE is the new interface to SCALE continuous energy physics.

14.19.1 Sub-databases

Name and type	Frequency
splice = { *ampx*}	Zero or more

14.19.2 Parameters

name
Label for the physics.

Default ‘sce’

Type string

`ce_lib_path`
Path to SCALE CE Library XML file.

Type file path for reading (extension ‘.xml’)

`ce_cache_path`
HDF5 file path for caching CE data for this problem.

Default

‘’

Type file path to read or modify (extension ‘.h5’) (empty value allowed)

`mode`
Particle transport mode.

Type particle type (‘p’,’n’,’np’,’neutron’,’photon’)

`comp_hdf5_output / comp_out`
Write compositions to the given HDF5 file, or skip if empty.

Default ‘compositions.h5’

Type file path to write (extension ‘.h5’) (empty value allowed)

`use_probability_tables / ptab`
Use probability table method for unresolved resonances.

Default True

Type boolean

`otf_elastic_scattering`
Use on-the-fly elastic scattering rather than table lookups.

Default False

Type boolean

`n_energy_min / n_emin`
Minimum global neutron energy cutoff for cross sections.

Default 1e-05

Type positive real number

Units eV

Applicability when ‘mode’ is ‘np’ or ‘n’.

`n_energy_max / n_emax`
Maximum global neutron_energy cutoff for cross sections.

Default 20000000.0

Type positive real number

Units eV

Applicability when ‘mode’ is ‘np’ or ‘n’.

`p_energy_min / p_emin`
Minimum global photon energy cutoff for cross sections.

Default 10000.0

Type positive real number

Units eV

Applicability when ‘mode’ is ‘np’ or ‘p’.

`p_energy_max / p_emax`
Maximum global photon energy cutoff for cross sections.

Default 25000000.0

Type positive real number

Units eV

Applicability when ‘mode’ is ‘np’ or ‘p’.

`thermal_energy_cutoff`
Thermalization energy cutoff for scattering kernels.

Default 10.0

Type positive real number

Units eV

`orig_zaid_n`
Replace CE data for these nuclides in problem materials.

The ZAID remapping options allow a ZAID present in the problem input to be replaced with a different ZAID. These options are most useful when entered in column input form:

```
orig_zaid_n : subs_zaid_n
       1001       8001001
      11022         11023
```

This is usually necessary when nuclide metadata is not available in the SCALE standard composition library.

Default []

Type list of nuclide specifier (e.g. U-235, 92235, u235, u-235m1)

`subs_zaid_n`
Substitute ZAID corresponding to ‘orig_zaid_n’.

Default []

Type list of nuclide specifier (e.g. U-235, 92235, u235, u-235m1)

`orig_zaid_p`
Replace photon CE data for these nuclides in problem materials.

Default []

Type list of nuclide specifier (e.g. U-235, 92235, u235, u-235m1)

`subs_zaid_p`
Substitute ZAID corresponding to ‘orig_zaid_p’.

Default []

Type list of nuclide specifier (e.g. U-235, 92235, u235, u-235m1)

14.19.3 Advanced parameters

These parameters are not meant for typical use.

`use_unionized_energy_grid`
Create unionized energy grid.

Default False

Type boolean

`load_gamma_production_data`
Load the gamma production data.

Type boolean

`use_collision_probabilities`
Use SCALE pre-calculated reaction probabilities.

Default False

Type boolean

`load_nubar`
Load average neutron production data.

Default True

Type boolean

`reactions`
Reactions to load (AMPX_MT values).

Default []

Type list of MT number or name (e.g. N_GAMMA, 102)

14.19.4 Commands

These commands generate one or more parameters.

`energy_limits`
Expand into parameters `n_energy_min`, `n_energy_max`.

`ce_lib`
Set `ce_lib` to the given value using SCALE DATA resolution.

The SCALE `FileNameAliases.txt` file is used to resolve the data files. Current options are:

Path	Description
ce_v7.0_endf.xml	ENDF/B-VII.0
ce_v7.xml	“
ce_v7_endf.xml	“
ce_v7.1_endf.xml	ENDF/B-VII.1
ce.xml	“

`ampx_kerma`
Load KERMA factors from an AMPX library.

14.19.5 Additional defaults

- **Default mode to 'n' for kcode, or based on sources if** present. .
- For particle mode np, 'load_gamma_production_data' defaults to True

For particle mode _default, 'load_gamma_production_data' defaults to False.

14.19.6 Additional restrictions

- Kcode problems must be run in mode 'n'.

14.19.7 Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
ce_cache_path	db	ce_cache_path
ce_lib_path	db:ce	ce_lib_path
load_gamma_production_data	db:ce	gamma_production
load_nubar	db:ce	nubar
mode	db	mode
n_energy_max	db:ce	nemax
n_energy_min	db:ce	nemin
orig_zaid_n	db:ce	orig_zaid_n
orig_zaid_p	db:ce	orig_zaid_p
p_energy_max	db:ce	pemax
p_energy_min	db:ce	pemin
reactions	db:ce	reactions
subs_zaid_n	db:ce	subs_zaid_n
subs_zaid_p	db:ce	subs_zaid_p
thermal_energy_cutoff	db:ce	ethermal
use_collision_probabilities	db:ce	collision_probabilities
use_probability_tables	db:ce	probability_tables
use_unionized_energy_grid	db:ce	unionize_energy

14.19.8 [PHYSICS=sce][SPLICE=ampx]

Parameters

`mg_lib_path`
Path to AMPX library to load data.

Type file path for reading

`orig_zaid_n`
Replace MG data for these nuclides in problem materials.

Default []

Type list of nuclide specifier (e.g. U-235, 92235, u235, u-235m1)

`subs_zaid_n`
Substitute ZAID corresponding to 'orig_zaid_n'.

Default []

Type list of nuclide specifier (e.g. U-235, 92235, u235, u-235m1)

orig_zaid_p
Replace photon MG data for these nuclides in problem materials.

Default []

Type list of nuclide specifier (e.g. U-235, 92235, u235, u-235m1)

subs_zaid_p
Substitute ZAID corresponding to 'orig_zaid_p'.

Default []

Type list of nuclide specifier (e.g. U-235, 92235, u235, u-235m1)

Advanced parameters

These parameters are not meant for typical use.

reactions
Multigroup reactions to splice into the CE data (AMPX_MT values).

Default []

Type list of MT number or name (e.g. N_GAMMA, 102)

Commands

These commands generate one or more parameters.

mg_lib
Set `mg_lib` to the given value using SCALE DATA resolution.

Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
mg_lib_path		xs_library

14.20 [PHYSICS=smg]

The SCALE MG physics package by default uses SCALE to calculate infinite homogeneous medium cross sections for all materials. No self-shielding is used, so results will generally not have good accuracy. The default cross section processing is primarily intended for generating deterministic solutions for hybrid calculations.

14.20.1 Parameters

name
Label for the physics.

Default 'smg'

Type string

mg_lib_path
MG Library file.

Type file path for reading

downscatter
Indicates whether this problem has downscatter only.

Default False

Type boolean

pn_order
Scattering order of problem.

Default 0

Type zero or positive odd integer

comp_hdf5_output / comp_out
Write compositions to the given HDF5 file, or skip if empty.

Default 'compositions.h5'

Type file path to write (extension '.h5') (empty value allowed)

14.20.2 Commands

These commands generate one or more parameters.

mg_lib
Set `mg_lib` to the given value using SCALE DATA resolution.

14.20.3 Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
downscatter	db	downscatter
mg_lib_path	db	xs_library
pn_order	db	Pn_order

14.21 [DEPLETION]

Omnibus couples Shift and ORIGEN, a depletion/transmutation analysis code, using ORIGEN's new C++ API. This new in-memory API avoids the prohibitive cost (on HPC systems) of writing to disk between transport and depletion steps and also enables each depletion step to be performed on each depletable region in parallel. Shift uses the same approach as VESTA to obtain microscopic per-nuclide reaction rates. This approach tallies ultra-fine-group fluxes in each depletion region and then uses these fluxes to collapse the microscopic CE cross sections into one-group reaction rates. Shift then sends these reaction rates to ORIGEN for a depletion calculation.

Currently, Shift uses constant-power depletion with either forward Euler or a "middlestep" method. In order to optimize the parallel efficiency of the depletion calculation, the depletable regions on each block are distributed amongst the available processors in order to minimize the number of depletion solves performed on each core. After every core has calculated the new concentrations for its depletion regions, the results are broadcast to every other core on the block.

14.21.1 Parameters

`deplete_cells / cells`

Labels of cells to deplete instead of all fissionable cells.

By default, when depletion is enabled, all "depletable" cells will be tracked and depleted. (Depletable cells correspond to materials with fissionable materials, or if a `[COMP]` block is used, materials with the depletable flag set.) This parameter overrides this default: cells with the listed labels will have their materials depleted.

Note: When an MCNP geometry is used, this command additionaly turns multiple cells with the same material into "unique" materials.

Default []

Type list of integers

`boron_level`

Boron level for a given time interval.

If given, the `boron_level` parameter must have the same length as `burn_time`: each value corresponds to the boron level for a given time step.

Default []

Type list of positive floats

`borated_cells / mod_cells`

Cells containing borated moderator.

Default []

Type list of integers

`tracking_nuclides`

Nuclides that will be tracked for depletion.

Default []

Type list of nuclides

`verbose`

Flag to print depletion banner following a depletion calculation.

Default True

Type boolean

`hdf5_output / depl_out`

Name of the output file to write, or blank to disable output.

Default 'depletion.h5'

Type file path to write (extension '.h5') (empty value allowed)

`write_predictor_data / write_p`

Flag to write predictor data to an HDF5-formatted output file.

Default False

Type boolean

Applicability when writing HDF5 output.

`write_xs`

Whether to write the collapsed origen XS to the HDF5 file.

Default False

Type boolean

Applicability when writing HDF5 output.

`max_step`
Maximum time before automatically increasing the number of steps.

Default 400.0

Type non-negative real number

Units days

`burn_time / burn`
Burnup times for each input step.

Type list of non-negative floats

Units days

`decay_time / decay`
Decay times for each input step.

Default []

Type list of non-negative floats

Units days

`num_steps`
Number of steps to take for each depletion entry.

The `num_steps` parameter must have the same length as the `max_step`, `burn_time`, and `decay_time`. It is the number of constant-flux calculations per entry. Increasing the number will increase the accuracy of the answer (by having better approximations of the depleted concentrations during the transport step) but will increase the computational cost (because more transport calculations must be performed).

Default []

Type list of positive integers

`constant_power_per_step / power`
Constant power to be applied per burnup step.

Default []

Type list of non-negative floats

Units MW

`constant_flux_per_step / flux`
Constant flux to be applied per burnup step.

Default []

Type list of non-negative floats

Units n/cm^2

`origen_library`
Filepath to an ORIGEN library file.

Default u'/usr/local/scale/data/origen_library/pwr.rev02.orglib'

Type file path for reading (extension '.orglib')

`max_burn_substep_size`
Maximum substep size for an ORIGEN time step of non-zero power/flux.

Default 40.0

Type non-negative real number

Units days

`max_decay_substep_size`
Maximum substep size for an ORIGEN time step of zero power/flux.

Default 75.0

Type non-negative real number

Units days

`nuclide_filter_type`
How to filter nuclides for transport calculations.

Default 'capture'

Type 'none', 'capture', or 'number_density'

`nuclide_filter_threshold`
Threshold at which nuclides below are removed from transport.

Default 0.05

Type non-negative real number

Applicability when 'nuclide_filter_type' is 'capture' or 'number_density'.

`depletion_solver`
Depletion solver type.

Default 'cram'

Type 'cram', or 'matrex'

`cram_order`
Order of the CRAM depletion solver.

Default 16

Type positive integer

Applicability when 'depletion_solver' is 'cram'.

`cram_internal_substeps`
Number of internal substeps in the CRAM depletion solver.

Default 1

Type positive integer

Applicability when 'depletion_solver' is 'cram'.

14.21.2 Commands

These commands generate one or more parameters.

`tracking_set`
append a set of TRITON nuclides to tracking_nuclides to track (none, addnux1, addnux-2, addnux2, addnux3, addnux4, all).

The `tracking_set` command exposes the TRITON `addnux` option to the user.

`tracking_set none` adds no extra nuclides to track (the default).

`tracking_set addnux1` corresponds to `addnux=1` and adds:

U-234, U-235, U-236, U-238, Np-237, Pu-238, Pu-239, Pu-240, Pu-241, Pu-242, Am-241, Am-242, Am-243, Cm-242, Cm-243

`tracking_set addnux-2` corresponds to `addnux=-2` and adds the above nuclides as well as:

H-1, B-10, B-11, N-14, O-16, Kr-83, Zr-94, Nb-93, Mo-95, Tc-99, Ru-106, Rh-103, Rh-105, Ag-109, Sn-126, I-135, Xe-131, Xe-135, Cs-133, Cs-134, Cs-135, Cs-137, Ce-144, Pr-143, Nd-143, Nd-145, Nd-146, Nd-147, Nd-148, Pm-147, Pm-148, Pm-149, Sm-147, Sm-149, Sm-150, Sm-151, Sm-152, Eu-151, Eu-153, Eu-154, Eu-155, Gd-152, Gd-154, Gd-155, Gd-156, Gd-157, Gd-158, Gd-160, Cm-244

`tracking_set addnux2` corresponds to `addnux=2` and adds the above nuclides as well as:

Zr-91, Zr-93, Zr-95, Zr-96, Nb-95, Mo-97, Mo-98, Mo-99, Mo-100, Ru-101, Ru-102, Ru-103, Ru-104, Pd-105, Pd-107, Pd-108, Cd-113, In-115, I-127, I-129, Xe-133, Ba-140, La-139, Ce-141, Ce-142, Ce-143, Pr-141, Nd-144, Sm-153, Eu-156

`tracking_set addnux3` corresponds to `addnux=3` and adds the above nuclides as well as:

Ge-72, Ge-73, Ge-74, Ge-76, As-75, Se-76, Se-77, Se-78, Se-80, Se-82, Br-79, Br-81, Kr-80, Kr-82, Kr-84, Kr-85, Kr-86, Rb-85, Rb-86, Rb-87, Sr-84, Sr-86, Sr-87, Sr-88, Sr-89, Sr-90, Y-89, Y-90, Y-91, Zr-90, Zr-92, Nb-94, Mo-92, Mo-94, Mo-96, Ru-96, Ru-98, Ru-99, Ru-100, Ru-105, Pd-102, Pd-104, Pd-106, Pd-110, Ag-107, Ag-111, Cd-106, Cd-108, Cd-110, Cd-111, Cd-112, Cd-114, Cd-115m, Cd-116, In-113, Sn-112, Sn-114, Sn-115, Sn-116, Sn-117, Sn-118, Sn-119, Sn-120, Sn-122, Sn-123, Sn-124, Sn-125, Sb-121, Sb-123, Sb-124, Sb-125, Sb-126, Te-120, Te-122, Te-123, Te-124, Te-125, Te-126, Te-127m, Te-128, Te-129m, Te-130, Te-132, I-130, I-131, Xe-124, Xe-126, Xe-128, Xe-129, Xe-130, Xe-132, Xe-134, Xe-136, Cs-136, Ba-134, Ba-135, Ba-136, Ba-137, Ba-138, La-140, Ce-140, Pr-142, Nd-142, Nd-150, Pm-151, Sm-144, Sm-148, Sm-154, Eu-152, Eu-157, Tb-159, Tb-160, Dy-160, Dy-161, Dy-162, Dy-163, Dy-164, Ho-165, Er-166, Er-167, Lu-175, Lu-176, Ta-181, W-182, W-183, W-184, W-186, Re-185, Re-187, Au-197, Th-230, Th-232, Pa-231, Pa-233, U-232, U-233

`tracking_set addnux4` corresponds to `addnux=4` and adds the above nuclides as well as:

H-2, H-3, He-3, He-4, Li-6, Li-7, Be-7, Be-9, N-15, O-17, F-19, Na-23, Mg-24, Mg-25, Mg-26, Al-27, Si-28, Si-29, Si-30, P-31, S-32, S-33, S-34, S-36, Cl-35, Cl-37, Ar-36, Ar-38, Ar-40, Ka-39, Ka-40, Ka-41, Ca-40, Ca-42, Ca-43, Ca-44, Ca-46, Ca-48, Sc-45, Ti-46, Ti-47, Ti-48, Ti-49, Ti-50, Cr-50, Cr-52, Cr-53, Cr-54, Mn-55, Fe-54, Fe-56, Fe-57, Fe-58, Co-58m, Co-58, Co-59, Ni-58, Ni-59, Ni-60, Ni-61, Ni-62, Ni-64, Cu-63, Cu-65, Ga-69, Ga-71, Ge-70, As-74, Se-74, Se-79, Kr-78, Ag-110m, Sn-113, Xe-123, Ba-130, Ba-132, Ba-133, La-138, Ce-136, Ce-138, Ce-139, Pm-148m, Gd-153, Dy-156, Dy-158, Ho-166m, Er-162, Er-164, Er-168, Er-170, Hf-174, Hf-176, Hf-177, Hf-178, Hf-179, Hf-180, Ta-182, Ir-191, Ir-193, Hg-196, Hg-198, Hg-199, Hg-200, Hg-201, Hg-202, Hg-204, Pb-204, Pb-206, Pb-207, Pb-208, Bi-209, Ra-223, Ra-224, Ra-225, Ra-226, Ac-225, Ac-226, Ac-227, Th-227, Th-228, Th-229, Th-233, Th-234, Pa-232, U-237, U-239, U-240, U-241, Np-235, Np-236, Np-238, Np-239, Pu-236, Pu-237, Pu-243, Pu-244, Pu-246, Am-242m, Am-244, Am-244m, Cm-241, Cm-245, Cm-246, Cm-247, Cm-248, Cm-249, Cm-250, Bk-249, Bk-250, Cf-249, Cf-250, Cf-251, Cf-252, Cf-253, Cf-254, Es-253, Es-254, Es-255

`tracking_set all` contains all the nuclides available in ORIGEN, which are the above nuclides as well as:

H-4, He-5, He-6, He-8, Li-8, Li-9, Be-8, Be-10, Be-11, Be-12, B-12, C-0, C-12, N-13, N-16, O-18, O-19, F-20, Ne-20, Ne-21, Ne-22, Ne-23, Na-22, Na-24, Na-24m, Na-25, Mg-27, Mg-28, Al-26, Al-28, Al-29, Al-30, Si-31, Si-32, P-32, P-33, P-34, S-25, S-35, S-37, Cl-36, Cl-38, Cl-38m, Ar-37, Ar-39, Ar-41, Ar-42, Ka-42, Ka-43, Ka-44, Ca-41, Ca-45, Ca-47, Ca-49, Sc-44, Sc-44m, Sc-45m, Sc-46, Sc-46m, Sc-47, Sc-48, Sc-49, Sc-50, Ti-44, Ti-45, Ti-51, V-48, V-49, V-50, V-51, V-52, V-53, V-54, Cr-48, Cr-49, Cr-51, Cr-55, Cr-66, Cr-67, Mn-52, Mn-53, Mn-54, Mn-56, Mn-57, Mn-58, Mn-66, Mn-67, Mn-68, Mn-69, Fe-55, Fe-59, Fe-60, Fe-65, Fe-66, Fe-67, Fe-68, Fe-69, Fe-70, Fe-71, Fe-72, Co-55, Co-56, Co-57, Co-60, Co-60m, Co-61, Co-62, Co-65, Co-66, Co-67, Co-68, Co-69, Co-70, Co-71, Co-72, Co-73, Co-74, Co-75, Ni-56, Ni-57, Ni-63, Ni-65, Ni-66, Ni-67, Ni-68, Ni-69, Ni-70, Ni-71, Ni-72, Ni-73, Ni-74, Ni-75, Ni-76, Ni-77, Ni-78, Cu-62, Cu-64, Cu-66, Cu-67, Cu-68, Cu-68m, Cu-69, Cu-70, Cu-70m, Cu-71, Cu-72, Cu-73, Cu-74, Cu-75, Cu-76, Cu-77, Cu-78, Cu-79, Cu-80, Cu-81, Zn-63, Zn-64, Zn-65, Zn-66, Zn-67, Zn-68, Zn-69, Zn-69m, Zn-70, Zn-71, Zn-71m, Zn-72, Zn-73, Zn-74, Zn-75, Zn-76, Zn-77, Zn-78, Zn-79, Zn-80, Zn-81, Zn-82, Zn-83, Ga-66, Ga-67, Ga-68, Ga-70, Ga-72, Ga-72m, Ga-73, Ga-74, Ga-74m, Ga-75, Ga-76, Ga-77, Ga-78, Ga-79, Ga-80, Ga-81, Ga-82, Ga-83, Ga-84, Ga-85, Ga-86, Ge-66, Ge-67, Ge-68, Ge-69, Ge-71, Ge-71m, Ge-73m, Ge-75, Ge-75m, Ge-77, Ge-77m, Ge-78, Ge-79, Ge-79m, Ge-80, Ge-81m, Ge-81, Ge-82, Ge-83, Ge-84, Ge-85, Ge-86, Ge-87, Ge-88, Ge-89, As-69, As-71, As-72, As-73, As-75m, As-76, As-77, As-78, As-79, As-80, As-81, As-82, As-82m, As-83, As-84, As-84m, As-85, As-86, As-87, As-88, As-89, As-90, As-91, As-92, Se-72, Se-73, Se-73m, Se-75, Se-77m, Se-79m, Se-81, Se-81m, Se-83m, Se-83, Se-84, Se-85, Se-86, Se-87, Se-88, Se-89, Se-90, Se-91, Se-92, Se-93, Se-94, Br-75, Br-76, Br-77, Br-77m, Br-78, Br-79m, Br-80, Br-80m, Br-82, Br-82m, Br-83, Br-84, Br-84m, Br-85, Br-86, Br-87, Br-88, Br-89, Br-90, Br-91, Br-92, Br-93, Br-94, Br-95, Br-96, Br-97, Kr-76, Kr-77, Kr-79m, Kr-79, Kr-81, Kr-81m, Kr-83m, Kr-85m, Kr-87, Kr-88, Kr-89, Kr-90, Kr-91, Kr-92, Kr-93, Kr-94, Kr-95, Kr-96, Kr-97, Kr-98, Kr-99, Kr-100, Rb-79, Rb-81, Rb-82, Rb-83, Rb-84, Rb-86m, Rb-88, Rb-89, Rb-90, Rb-90m, Rb-91, Rb-92, Rb-93, Rb-94, Rb-95, Rb-96, Rb-97, Rb-98, Rb-99, Rb-100, Rb-101, Rb-102, Sr-82, Sr-83, Sr-85, Sr-85m, Sr-87m, Sr-91, Sr-92, Sr-93, Sr-94, Sr-95, Sr-96, Sr-97, Sr-98, Sr-99, Sr-100, Sr-101, Sr-102, Sr-103, Sr-104, Sr-105, Y-85, Y-86, Y-87, Y-87m, Y-88, Y-89m, Y-90m, Y-91m, Y-92, Y-93, Y-93m, Y-94, Y-95, Y-96, Y-96m, Y-97, Y-97m, Y-98, Y-98m, Y-99, Y-100, Y-101, Y-102, Y-103, Y-104, Y-105, Y-106, Y-107, Y-108, Zr-86, Zr-87, Zr-88, Zr-89, Zr-89m, Zr-90m, Zr-97, Zr-98, Zr-99, Zr-100, Zr-101, Zr-102, Zr-103, Zr-104, Zr-105, Zr-106, Zr-107, Zr-108, Zr-109, Zr-110, Nb-89, Nb-90, Nb-91, Nb-91m, Nb-92, Nb-92m, Nb-93m, Nb-94m, Nb-95m, Nb-96, Nb-97, Nb-97m, Nb-98, Nb-98m, Nb-99, Nb-99m, Nb-100, Nb-100m, Nb-101, Nb-102, Nb-102m, Nb-103, Nb-104, Nb-104m, Nb-105, Nb-106, Nb-107, Nb-108, Nb-109, Nb-110, Nb-111, Nb-112, Nb-113, Mo-90, Mo-91, Mo-93, Mo-93m, Mo-101, Mo-102, Mo-103, Mo-104, Mo-105, Mo-106, Mo-107, Mo-108, Mo-109, Mo-110, Mo-111, Mo-112, Mo-113, Mo-114, Mo-115, Tc-93, Tc-95, Tc-95m, Tc-96, Tc-97, Tc-97m, Tc-98, Tc-99m, Tc-100, Tc-101, Tc-102, Tc-102m, Tc-103, Tc-104, Tc-105, Tc-106, Tc-107, Tc-108, Tc-109, Tc-110, Tc-111, Tc-112, Tc-113, Tc-114, Tc-115, Tc-116, Tc-117, Tc-118, Ru-95, Ru-97, Ru-107, Ru-108, Ru-109, Ru-109m, Ru-110, Ru-111, Ru-112, Ru-113, Ru-114, Ru-115, Ru-116, Ru-117, Ru-118, Ru-119, Ru-120, Rh-99, Rh-99m, Rh-100, Rh-101, Rh-101m, Rh-102, Rh-102m, Rh-103m, Rh-104, Rh-104m, Rh-105m, Rh-106, Rh-106m, Rh-107, Rh-108, Rh-108m, Rh-109, Rh-109m, Rh-110, Rh-110m, Rh-111, Rh-112, Rh-113, Rh-114, Rh-115, Rh-116, Rh-117, Rh-118, Rh-119, Rh-120, Rh-121, Rh-122, Rh-123, Pd-99, Pd-100, Pd-101, Pd-103, Pd-107m, Pd-109, Pd-109m, Pd-111, Pd-111m, Pd-112, Pd-113, Pd-114, Pd-115, Pd-116, Pd-117, Pd-118, Pd-119, Pd-120, Pd-121, Pd-122, Pd-123, Pd-124, Pd-125, Pd-126, Ag-103, Ag-105, Ag-105m, Ag-106, Ag-106m, Ag-107m, Ag-108, Ag-108m, Ag-109m, Ag-110, Ag-111m, Ag-112, Ag-113, Ag-113m, Ag-114, Ag-115, Ag-115m, Ag-116, Ag-116m, Ag-117, Ag-117m, Ag-118, Ag-118m, Ag-119, Ag-120, Ag-120m, Ag-121, Ag-122, Ag-122m, Ag-123, Ag-124, Ag-125, Ag-126, Ag-127, Ag-128, Ag-129, Ag-130, Cd-105, Cd-107, Cd-109, Cd-111m, Cd-113m, Cd-115, Cd-117, Cd-117m, Cd-118, Cd-119, Cd-119m, Cd-120, Cd-121, Cd-121m, Cd-122, Cd-123, Cd-123m, Cd-124, Cd-125, Cd-126, Cd-127, Cd-128, Cd-129, Cd-130, Cd-131, Cd-132, In-107, In-109, In-111m, In-111, In-112, In-112m, In-113m, In-114m, In-114, In-115m, In-116m, In-116, In-117m, In-117, In-118m, In-118, In-119m, In-119, In-120m, In-120, In-121m, In-121, In-122m, In-122, In-123m, In-123, In-124m, In-124, In-125m, In-125, In-126m, In-126, In-127m, In-127, In-128m, In-128, In-129m, In-129, In-130m, In-130, In-131m, In-131, In-132, In-133, In-134, In-135, Sn-111, Sn-113m, Sn-117m, Sn-119m, Sn-121, Sn-121m, Sn-123m, Sn-125m, Sn-127, Sn-127m, Sn-128, Sn-128m, Sn-129, Sn-129m, Sn-130, Sn-130m, Sn-131, Sn-131m, Sn-132, Sn-133, Sn-134, Sn-135, Sn-136, Sn-137, Sb-113, Sb-115m, Sb-115, Sb-117, Sb-118m, Sb-118, Sb-119, Sb-120m, Sb-120, Sb-122m, Sb-122, Sb-124m, Sb-126m, Sb-127, Sb-128m, Sb-128, Sb-129, Sb-130m, Sb-130, Sb-131, Sb-132m, Sb-132, Sb-133, Sb-134m, Sb-134, Sb-135, Sb-136, Sb-137, Sb-138, Sb-139, Te-115, Te-117, Te-118, Te-119m, Te-119, Te-121, Te-121m, Te-123m, Te-125m, Te-127, Te-129, Te-131, Te-131m, Te-133, Te-133m, Te-134, Te-135, Te-136, Te-137, Te-138, Te-139, Te-140, Te-141, Te-142, I-121, I-122, I-123, I-124, I-125, I-126, I-128, I-130m, I-132, I-132m, I-133, I-133m, I-134m, I-134, I-136m, I-136, I-137, I-138, I-139, I-140, I-141, I-142, I-143, I-144, I-145, Xe-122, Xe-125, Xe-125m, Xe-127m, Xe-127, Xe-129m, Xe-131m, Xe-133m, Xe-134m, Xe-135m, Xe-137, Xe-138, Xe-139, Xe-140, Xe-141, Xe-142, Xe-143, Xe-144, Xe-145, Xe-145m, Xe-146, Xe-147, Cs-127, Cs-128, Cs-129, Cs-130, Cs-131, Cs-132, Cs-134m, Cs-135m, Cs-136m, Cs-138m, Cs-138, Cs-139, Cs-140, Cs-141, Cs-142, Cs-143, Cs-144, Cs-145, Cs-146, Cs-147, Cs-148, Cs-149, Cs-150, Cs-151, Ba-128, Ba-129, Ba-131, Ba-131m, Ba-133m, Ba-135m, Ba-136m, Ba-137m, Ba-139, Ba-141, Ba-142, Ba-143, Ba-144, Ba-145, Ba-146, Ba-147, Ba-148, Ba-149, Ba-150, Ba-151, Ba-152, Ba-153, La-133, La-134, La-135, La-136, La-137, La-141, La-142, La-143, La-144, La-145, La-146m, La-146, La-147, La-148, La-149, La-150, La-151, La-152, La-153, La-154, La-155, Ce-134, Ce-135, Ce-137m, Ce-137, Ce-139m, Ce-145, Ce-146, Ce-147, Ce-148, Ce-149, Ce-150, Ce-151, Ce-152, Ce-153, Ce-154, Ce-155, Ce-156, Ce-157, Pr-139, Pr-140, Pr-142m, Pr-144m, Pr-144, Pr-145, Pr-146, Pr-147, Pr-148m, Pr-148, Pr-149, Pr-150, Pr-151, Pr-152, Pr-153, Pr-154, Pr-155, Pr-156, Pr-157, Pr-158, Pr-159, Nd-140, Nd-141, Nd-149, Nd-151, Nd-152, Nd-153, Nd-154, Nd-155, Nd-156, Nd-157, Nd-158, Nd-159, Nd-160, Nd-161, Pm-141, Pm-143, Pm-144, Pm-145, Pm-146, Pm-150, Pm-152m, Pm-152, Pm-153, Pm-154m, Pm-154, Pm-155, Pm-156, Pm-157, Pm-158, Pm-159, Pm-160, Pm-161, Pm-162, Pm-163, Sm-143, Sm-143m, Sm-145, Sm-146, Sm-155, Sm-156, Sm-157, Sm-158, Sm-159, Sm-160, Sm-161, Sm-162, Sm-163, Sm-164, Sm-165, Eu-145, Eu-146, Eu-147, Eu-148, Eu-149, Eu-150m, Eu-150, Eu-152m, Eu-154m, Eu-158, Eu-159, Eu-160, Eu-161, Eu-162, Eu-163, Eu-164, Eu-165, Eu-166, Eu-167, Gd-146, Gd-147, Gd-148, Gd-149, Gd-150, Gd-151, Gd-153m, Gd-155m, Gd-159, Gd-161, Gd-162, Gd-163, Gd-164, Gd-165, Gd-166, Gd-167, Gd-168, Gd-169, Tb-151, Tb-152, Tb-153, Tb-154m, Tb-154, Tb-155, Tb-156m, Tb-156, Tb-157, Tb-158m, Tb-158, Tb-161, Tb-162, Tb-163, Tb-164, Tb-165, Tb-166, Tb-167, Tb-168, Tb-169, Tb-170, Tb-171, Dy-154, Dy-155, Dy-157, Dy-159, Dy-165, Dy-165m, Dy-166, Dy-167, Dy-168, Dy-169, Dy-170, Dy-171, Dy-172, Ho-159m, Ho-159, Ho-160m, Ho-160, Ho-161m, Ho-161, Ho-162m, Ho-162, Ho-163m, Ho-163, Ho-164m, Ho-164, Ho-166, Ho-167, Ho-168, Ho-169, Ho-170m, Ho-170, Ho-171, Ho-172, Er-160, Er-161, Er-163, Er-165, Er-167m, Er-169, Er-171, Er-172, Tm-165, Tm-166, Tm-167, Tm-168, Tm-169, Tm-170m, Tm-170, Tm-171, Tm-172, Tm-173, Yb-166, Yb-167, Yb-168, Yb-169m, Yb-169, Yb-170, Yb-171, Yb-172, Yb-173, Yb-174, Yb-175m, Yb-175, Yb-176, Yb-177, Lu-169m, Lu-169, Lu-170, Lu-171m, Lu-171, Lu-172m, Lu-172, Lu-173, Lu-174m, Lu-174, Lu-176m, Lu-177m, Lu-177, Hf-170, Hf-171, Hf-172, Hf-173, Hf-175, Hf-178m, Hf-179m, Hf-180m, Hf-181, Hf-182, Ta-177, Ta-178, Ta-179, Ta-180m, Ta-180, Ta-182m, Ta-183, W-178, W-180, W-181, W-183m, W-185, W-185m, W-187, W-188, W-189, Re-181, Re-182, Re-182m, Re-183, Re-184, Re-184m, Re-186, Re-186m, Re-188, Re-188m, Re-189, Os-182, Os-183, Os-184, Os-185, Os-186, Os-187, Os-188, Os-189, Os-189m, Os-190, Os-190m, Os-191, Os-191m, Os-192, Os-193, Os-194, Ir-185, Ir-186, Ir-188, Ir-189, Ir-189m, Ir-190, Ir-191m, Ir-192, Ir-192m, Ir-193m, Ir-194, Ir-194m, Ir-196, Ir-196m, Pt-188, Pt-189, Pt-190, Pt-191, Pt-192, Pt-193, Pt-193m, Pt-194, Pt-195, Pt-195m, Pt-196, Pt-197, Pt-197m, Pt-198, Pt-199, Pt-199m, Pt-200, Au-193, Au-194, Au-195, Au-195m, Au-196, Au-198, Au-198m, Au-199, Au-199m, Au-200, Au-200m, Hg-193m, Hg-193, Hg-194, Hg-195m, Hg-195, Hg-197, Hg-197m, Hg-199m, Hg-203, Hg-205, Hg-206, Tl-200, Tl-201, Tl-202, Tl-203, Tl-204, Tl-205, Tl-206, Tl-207, Tl-208, Tl-209, Tl-210, Pb-200, Pb-202, Pb-203, Pb-205, Pb-205m, Pb-207m, Pb-209, Pb-210, Pb-211, Pb-212, Pb-214, Bi-205, Bi-206, Bi-207, Bi-208, Bi-210, Bi-210m, Bi-211, Bi-212, Bi-212m, Bi-213, Bi-214, Po-206, Po-207, Po-208, Po-209, Po-210, Po-211, Po-211m, Po-212, Po-213, Po-214, Po-215, Po-216, Po-218, At-216, At-217, At-218, Rn-216, Rn-217, Rn-218, Rn-219, Rn-220, Rn-222, Fr-220, Fr-221, Fr-222, Fr-223, Ra-220, Ra-222, Ra-227, Ra-228, Ac-224, Ac-228, Th-226, Th-231, Pa-228, Pa-229, Pa-230, Pa-234, Pa-234m, Pa-235, U-230, U-231, Np-234, Np-236m, Np-240, Np-240m, Np-241, Pu-237m, Pu-245, Pu-247, Am-239, Am-240, Am-245, Am-246, Am-247, Cm-240, Cm-251, Bk-245, Bk-246, Bk-247, Bk-248, Bk-248m, Bk-251, Cf-246, Cf-248, Cf-255, Es-251, Es-252, Es-254m

These tables are available in the SCALE 6.1 manual.

14.21.3 Additional restrictions

- Nuclides in 'tracking_nuclides' are validated against the TRITON nuclide list..

14.21.4 Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
burn_time	db	burn_steps
constant_flux_per_step	db	fluxes
constant_power_per_step	db	powers
cram_internal_substeps	db	cramInternalSubsteps
cram_order	db	cramOrder
decay_time	db	decay_steps
depletion_solver	db	origen_solver
hdf5_output	db	hdf5_filename
max_burn_substep_size	db	max_burn_step_size
max_decay_substep_size	db	max_decay_step_size
max_step	db	max_burn_length
nuclide_filter_threshold	db	nuclide_threshold
nuclide_filter_type	db	nuclide_threshing
num_steps	db	num_libs
origen_library	db	library
tracking_nuclides	db	tracking_nuclides
verbose	db	depletion_diagnostics
write_predictor_data	db	write_predictor
write_xs	db	write_xs

14.22 [SHIFT]

The Shift database is for Monte Carlo execution parameters. It is broken into a top-level database and three subsidiary databases. If a KCODE problem is being run (see *[PROBLEM]*), the [KCODE] subdatabase is required. It specifies the number of inactive and active cycles, as well as the particles per cycle. If running a fixed-source problem, the `num_histories` parameter in the main [SHIFT] database is required.

Shift supports experimental domain decomposition for very large problems. If desired, the *[DECOMPOSITION]* block should be included to specify the spatial decomposition.

14.22.1 Sub-databases

Name and type	Frequency
decomposition	Exactly once
kcode	Optional
vr	Exactly once

14.22.2 Parameters

`do_transport`
If false, this disables the actual solve.

Setting `do_transport` to `false` is a way to ensure problem integrity without running an expensive transport calculation. It disables transport itself but allows the rest of the code (including buliding sources, tallies, and physics) to run. This is simliar to `parm=check` in SCALE.

Changed in version 5.3: This replaces the `mode check` problem option.

Default True

Type boolean

`num_histories / np`
Number of histories.

Default 1000

Type positive integer

Applicability when problem mode is 'forward' or 'adjoint'.

`num_dd_samples`
Number of test samples to determine initial particle balance.

With domain decomposed problems, an *a priori* determination of the particle balance across blocks is performed before transport. Each domain randomly samples the source for a total of `num_dd_samples` times; the fraction of source particles found within the "core" (non-overlapping) component of each domain is used to determine the number of particles emitted from the source on that domain.

Type integer

Applicability when Shift spatial partitioning is domain decomposed.

`output_fraction_completed`
Output fraction of completed particles.

Default 1.0

Type positive real number

`check_frequency / chk_freq`
Check frequency for domain decomposed current cycle completion.

Default 1

Type positive integer

Applicability when Shift spatial partitioning is domain decomposed.

particle_buffer_size / bufsize
Size of particle buffer for DD problem.

Default 1000

Type positive integer

Applicability when Shift spatial partitioning is domain decomposed.

14.22.3 Advanced parameters

These parameters are not meant for typical use.

physics
Name of the physics DB to use.

Type string

14.22.4 Additional defaults

- 'physics' defaults to the name of the last PHYSICS entered..
- Domain decomposition samples (num_dd_samples) defaults to num_histories / 10.
- A **[DECOMPOSITION]** subdatabase is added by default.
- A **[VR]** subdatabase is added by default.

14.22.5 Additional restrictions

- A [KCODE] sub-database must be included for eigenvalue problems.

14.22.6 Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
check_frequency		mc_check_freq
num_dd_samples	source_db	dd_test_samples
num_histories	source_db	Np
output_fraction_completed		mc_diag_frac

14.22.7 [SHIFT][DECOMPOSITION]

Parameters

x_partition / x
Boundary mesh along the X axis.

Default [-1000000000000.0, 1000000000000.0]

Type list of two or more monotonically increasing floats

y_partition / y
Boundary mesh along the Y axis.

Default [-1000000000000.0, 1000000000000.0]

Type list of two or more monotonically increasing floats

`z_partition / z`
Boundary mesh along the Z axis.

Default [-1000000000000.0, 1000000000000.0]

Type list of two or more monotonically increasing floats

`overlap`
Fraction of DD domain overlap.

Default 0.0

Type real number inclusive [0.0, 1.0]

Applicability when Shift spatial partitioning is domain decomposed.

Advanced parameters

These parameters are not meant for typical use.

`boundary_reflect`
Whether to reflect on each side of the boundary mesh.

Default [0, 0, 0, 0, 0, 0]

Type list of six zero/one integers for -X,+X,-Y,+Y,-Z,+Z

Additional restrictions

- The number of processors must divide evenly into the number of decompositions if a [RUN] block is used.

Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
boundary_reflect	boundary_db	reflect_bnd_mesh
overlap	boundary_db	overlap
x_partition	boundary_db	x_bnd_mesh
y_partition	boundary_db	y_bnd_mesh
z_partition	boundary_db	z_bnd_mesh

14.22.8 [SHIFT][KCODE]

Parameters

`num_histories_per_cycle / npk`
Number of histories per cycle.

Default 1000

Type positive integer

`x_entropy`
Boundaries in x for the Shannon entropy mesh.

Type list of two or more monotonically increasing floats

Units cm

`y_entropy`
Boundaries in y for the Shannon entropy mesh.

Type list of two or more monotonically increasing floats

Units cm

`z_entropy`
Boundaries in z for the Shannon entropy mesh.

Type list of two or more monotonically increasing floats

Units cm

`initial_keff / initk`
Initial keff value.

Default 1.0

Type positive real number

`num_cycles / nk`
Number of kcode cycles.

Default 50

Type integer

`num_inactive_cycles / nik`
Number of inactive kcode cycles.

Default 10

Type integer

`quiet_cycle_output / quiet`
If true, do not output kcycle output to stdin.

Default False

Type boolean

Additional defaults

- Shannon entropy defaults to partition bounds.

Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
initial_keff	solver_db	keff_init
num_cycles	solver_db	num_cycles
num_histories_per_cycle	source_db	Np
num_inactive_cycles	solver_db	num_inactive_cycles
quiet_cycle_output	solver_db	quiet
x_entropy	solver_db	entropy_x
y_entropy	solver_db	entropy_y
z_entropy	solver_db	entropy_z

14.22.9 [SHIFT][VR]

Parameters

`variance_reduction / vr`
Variance reduction method.

Default 'roulette'

Type 'ww', 'analog', or 'roulette'

`weight_cutoff / wc`
Particle weight cutoff for roulette.

Default 0.25

Type non-negative real number

Applicability when 'variance_reduction' is 'roulette' or 'ww'.

`weight_survival / ws`
Particle weight survival for roulette.

Default 0.5

Type non-negative real number

Applicability when 'variance_reduction' is 'roulette'.

`ww_lower_factor / wwlow`
Lower weight window ratio.

Default 0.5

Type real number inside (0, 1)

Applicability when 'variance_reduction' is 'ww'.

`ww_upper_factor / wwhigh`
Upper weight window ratio.

Default 2.5

Type real number greater than 1

Applicability when 'variance_reduction' is 'ww'.

Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
variance_reduction	vr_db	method
weight_cutoff	vr_db	weight_cutoff
weight_survival	vr_db	weight_survival
ww_lower_factor	vr_db	ww_lower_factor
ww_upper_factor	vr_db	ww_upper_factor

14.23 [DENOVO]

14.23.1 Parameters

`do_transport`
If false, this disables the actual solve.

Default True

Type boolean

`method`
Solution method or spatial discretization.

Default 'spn'

Type 'spn', 'sc', 'ld', or 'tld'

`x`
Mesh coordinates along the X axis.

Type list of monotonically increasing floats

`y`
Mesh coordinates along the Y axis.

Type list of monotonically increasing floats

`z`
Mesh coordinates along the Z axis.

Type list of monotonically increasing floats

`boundary`
Boundary conditions on the deterministic problem.

Default 'vacuum'

Type 'vacuum', 'isotropic', or 'reflect'

`boundary_reflect`
Whether to reflect on each side of the problem.

Default [1, 1, 1, 1, 1, 1]

Type list of six zero/one integers for -X,+X,-Y,+Y,-Z,+Z

Applicability when 'boundary' is 'reflect'.

`physics`
Name of the associated physics database.

Type string

`quadrature`
Discrete ordinates quadrature set class.

> **Default** ‘qr’
>
> **Type** ‘levelsym’, ‘glproduct’, ‘qr’, ‘ldfe’, or ‘userdefined’
>
> **Applicability** when Denovo is using a discrete ordinates (SN) solver.

`quad_order`
Level-symmetric quadrature set order.

> **Default** 10
>
> **Type** non-negative integer
>
> **Applicability** when ‘quadrature’ is ‘qr’ or ‘levelsym’.

`quad_num_azi`
Number of azimuthal angles per level per octant.

> **Default** 4
>
> **Type** positive integer
>
> **Applicability** when ‘quadrature’ is ‘ldfe’ or ‘qr’ or ‘glproduct’.

`quad_num_azi_vec`
List of the number of azimuthal angles per polar angle per octant, ordered from pole to equator.

> **Default** []
>
> **Type** list of positive integers
>
> **Applicability** when ‘quadrature’ is ‘qr’.

`quad_num_polar`
Number of polar angles per level per octant.

> **Default** 4
>
> **Type** positive integer
>
> **Applicability** when ‘quadrature’ is ‘ldfe’ or ‘qr’ or ‘glproduct’.

`quad_file`
User-specified quadrature set file.

> **Type** file path for reading
>
> **Applicability** when ‘quadrature’ is ‘userdefined’.

`ldfe_order`
Order for the LDFE quadrature set.

> **Default** 1
>
> **Type** positive integer
>
> **Applicability** when ‘quadrature’ is ‘ldfe’.

`quad_polar_axis`
Axis of rotation for product quadrature sets.

> **Default** ‘z’
>
> **Type** ‘x’, ‘y’, or ‘z’
>
> **Applicability** when ‘quadrature’ is ‘qr’ or ‘glproduct’.

`x_blocks`
The number of spatial partitions along the x axis.

Type non-negative integer

`y_blocks`
The number of spatial partitions along the y axis.

Type non-negative integer

`z_blocks`
The number of pipelining blocks along the z axis.

Type non-negative integer

`energy_sets`
The number of energy partitions.

Default 1

Type non-negative integer

Applicability when Upscatter is enabled.

`partition_upscatter`
Partition just the upscatter groups.

Default True

Type boolean

Applicability when Denovo energy partitioning is multiset.

`first_group`
The first energy group to solve.

Default 0

Type non-negative integer

`last_group`
The last energy group to solve.

Default 1000000

Type non-negative integer

`tolerance`
Convergence criterion for the deterministic solver.

Default 0.001

Type real number inside (0, 1)

`max_iterations`
The maximum number of Krylov or source iterations.

Default 100

Type non-negative integer

`krylov_space`
The number of Krylov vectors to store when using GMRES.

Default 20

Type integer

`multigroup_solver`
Solution technique for the multigroup block solve.

Default ‘gauss_seidel’

Type ‘gauss_seidel’, or ‘krylov’

Applicability when Upscatter is enabled.

`solver`
Solver used in each within-group SN solve.

Default ‘gmres’

Type ‘gmres’, ‘si’, or ‘gmres_r’

`two_grid`
Enable two-grid upscattering acceleration.

Default False

Type boolean

Applicability when Upscatter is enabled.

`upscatter_solver`
Within-group solver for upscattering groups.

Default ‘gmres’

Type ‘gmres’, or ‘si’

Applicability when ‘multigroup_solver’ is ‘gauss_seidel’.

`upscatter_tolerance`
Convergence criterion for the upscatter iterations.

Default 0.01

Type real number inside (0, 1)

Applicability when ‘multigroup_solver’ is ‘gauss_seidel’.

`upscatter_inner_iterations`
Maximum number of iterations for the within-group upscattering solve.

Default 10

Type non-negative integer

Applicability when ‘multigroup_solver’ is ‘gauss_seidel’.

`upscatter_inner_tolerance`
Convergence criterion for the within-group upscattering solve.

Default 0.01

Type real number inside (0, 1)

Applicability when ‘multigroup_solver’ is ‘gauss_seidel’.

`mix_tolerance`
Tolerance for collapsing similar mixed materials into one.

Default 0.05

Type real number inside (0, 1)

`rays_deterministic`
If true, use face midpoints rather than stratified sampling.

Default False

Type boolean

`rays_per_face`
Number of ray trace rays to be fired per mesh face.

Default 4

Type positive square integer

`raytrace_axes`
Axis/axes along which to fire rays for ray trace.

Default ‘xyz’

Type axis or axes (‘x’,’zy’,’xyz’)

`silo_mixing_table`
Write mixed materials to Silo output.

Default True

Type boolean

14.23.2 Advanced parameters

These parameters are not meant for typical use.

`downscatter`
Whether to disable upscattering.

Type boolean

`pn_order`
Order of the Legendre scattering expansion.

Type non-negative integer

`use_cuda`
Use the CUDA sweeper to solve the KBA equations (experimental).

Default False

Type boolean

Applicability when ‘method’ is ‘sc’ or ‘ld’.

`void_matid`
Material ID to be used when raytrace is outside the problem.

Default 0

Type integer

14.23.3 Additional defaults

- ‘physics’ defaults to the name of the last PHYSICS entered..
- Auto-set multigroup physics parameters.
- If a [RUN] block is present, come up with a logically square KBA decomposition.
- Force number of Z blocks in SPN to 1, and default to nx * ny for SN.

14.23.4 Additional restrictions

- The number of processors must be equal to the decomposition size (X * Y * E) if a [RUN] block is used..
- Extra parameter validation.

14.23.5 Advanced

The following parameters are renamed or reorganized when being output to the Omnibus XML input file:

Parameter	Renamed subdatabase	Renamed parameter
boundary_reflect	boundary_db	reflect
energy_sets		num_sets
krylov_space		aztec_kspace
ldfe_order	quadrature_db	order
max_iterations		max_itr
mix_tolerance	raytrace_db	mix_tolerance
multigroup_solver		mg_solver
partition_upscatter		partition_upscatter
pn_order		Pn_order
quad_file	quadrature_db	quadrature_file_name
quad_num_azi	quadrature_db	azimuthals_octant
quad_num_azi_vec	quadrature_db	azimuthals_vector
quad_num_polar	quadrature_db	polars_octant
quad_order	quadrature_db	Sn_order
quad_polar_axis	quadrature_db	polar_axis
quadrature	quadrature_db	quad_type
rays_deterministic	raytrace_db	deterministic
rays_per_face	raytrace_db	rays_per_face
raytrace_axes	raytrace_db	axes
silo_mixing_table	silo_db	mixing_table
solver		within_group_solver
two_grid	upscatter_db	upscatter_acceleration
upscatter_inner_iterations	upscatter_db	inner_itr
upscatter_inner_tolerance	upscatter_db	inner_tolerance
upscatter_solver	upscatter_db	up_group_solver
upscatter_tolerance	upscatter_db	tolerance
void_matid	raytrace_db	void_matid
x		x_edges
x_blocks		num_blocks_i
y		y_edges
y_blocks		num_blocks_j
z		z_edges
z_blocks		num_z_blocks

14.24 [MANUALWW]

14.24.1 Parameters

`ww_file`
Manual weight window HDF5 file.

Type file path for reading (extension '.h5')

14.25 [RUN]

The [RUN] database enables support for running the Omnibus executable with an inline command `omnibus-run`. The auto-run feature will format and echo program output to the screen, and it automatically saves the output and error streams to disk.

If the `SCALE` and `DATA` environment variables are not set, `omnibus-run` will use the values determined at configuration time.

14.26 [RUN=serial]

Run as a serial process on the local machine, echoing output to the user. If the `omnibus-run` process is aborted, the `omnibus` command will also abort.

14.26.1 Advanced parameters

These parameters are not meant for typical use.

`omnibus`
Path to the Omnibus executable.

Default '/SCALE/bin/omnibus'

Type file path for reading

14.27 [RUN=mpi]

Run as an MPI process on the local machine, echoing output to the user. If the `omnibus-run` process is aborted, the `mpirun omnibus` command will also abort.

14.27.1 Parameters

`np`
Number of processors to run.

Type positive integer

`mpiexec_args`
Optional arguments passed to mpiexec..

Default []

Type list of strings

`machinefile`
Optional MPI machinefile for running on multiple nodes..

Default

''

Type file path for reading (empty value allowed)

14.27.2 Advanced parameters

These parameters are not meant for typical use.

mpiexec
Path to the MPI run command.

Default u'/opt/gcc48/openmpi/bin/mpiexec'

Type file path for reading

npflag
Number of processors flag.

Default u'-np'

Type string

omnibus
Path to the Omnibus executable.

Default '/SCALE/bin/omnibus'

Type file path for reading

14.28 [RUN=pbs]

Create a PBS run file for this job. A typical PBS run block will look like:

```
[RUN=pbs]
nodes    1
ppn      16
pmem     7900mb
walltime "24:00:00"
```

If the `omnibus-run` command is aborted while the job is run, the job **will not** be automatically aborted. You must run `qdel` separately to abort the job.

14.28.1 Parameters

nodes
Number of nodes to use.

Type positive integer

ppn
Number of processors per node.

Type positive integer

pmem
Amount of memory per processor (e.g. '7900mb').

Default

''

Type string

`walltime`
Wall time limit.

Note: It is common to have colons as part of the wall time; since colons must be escaped in Omnibus ASCII input, you will almost always need to escape the wall time input parameter:

```
[RUN=pbs]
walltime "24:00:00"
```

Default
‘’

Type string

`queue`
Queue to use.

Default
‘’

Type string

`join`
Output joining flags.

Default ‘oe’

Type string

`email`
Email address of recipient.

Type string

`when_email`
When to email.

Default ‘ea’

Type string

`qsub`
PBS submission command or path.

Default ‘qsub’

Type string

`qstat`
PBS status command or path.

Default ‘qstat’

Type string

`qdel`
PBS deletion command or path.

Default ‘qdel’

Type string

`mpiexec_args`
Optional arguments passed to mpiexec..

Default []

Type list of strings

`detach`
Whether to simply submit the job and to not follow it.

Default False

Type boolean

14.28.2 Advanced parameters

These parameters are not meant for typical use.

`mpiexec`
Path to the MPI run command.

Default u'/opt/gcc48/openmpi/bin/mpiexec'

Type file path for reading

`omnibus`
Path to the Omnibus executable.

Default '/SCALE/bin/omnibus'

Type file path for reading

14.28.3 Additional defaults

- Email address defaults to `git config author.email`.
- Automatically set MPI arguments and processors for clusters.

CHAPTER

FIFTEEN

POSTPROCESSING

In order to postprocess HDF5 data with the Python front-end, it's necessary to download h5py, which is often easily installed with package managers such as MacPorts.

15.1 Cylindrical tally postprocessing

To load the HDF5 database into Python:

```
>>> from omnibus.postprocess import cyltally
>>> tallies = cyltally.load('cyltally.h5')
INFO: Loading HDF5 from cyltally.h5
```

The result is a dictionary of tallies, each corresponding to a `CYLTALLY` entry in the Omnibus input, with the corresponding `name` attribute:

```
>>> tallies.keys()
['n:SVXF:VXF-3', 'n:TRRH:FT-E7']
>>> tal = tallies['n:SVXF:VXF-3']
>>> print tal
TallyResult: n:SVXF:VXF-3
```

The top-level tally has several attributes:

```
>>> print tal.description
neutron flux in SVXF target location VXF-3
>>> print tal.mesh.translation
[ -9.15368  38.12784   0.      ]
>>> tal.bin_bounds.num_total_groups
200
>>> tal.multipliers
(u'flux',)
```

And its data can be dumped to a multigroup Silo file for visualization:

```
>>> tal.write_to_silo()
INFO: Writing cylindrical tally 'n:SVXF:VXF-3' to n_SVXF_VXF_3.silo
```

The next layer of the tally hierarchy are the multipliers specified for each tally, which we provide a dictionary-like shortcut for accessing:

```
>>> print tal.tallies
{'flux': <TallyMultiplierResult ...>}
>>> rxntal = tal['flux']
>>> print rxntal
TallyMultiplierResult: flux
```

These have the actual flux results stored as a TallyContainer of mean and variance:

```
>>> rxntal.total
TallyContainer(mean=array([[[  7.93285055e-05]]]), var=array([[[
1.46720455e-12]]]))
```

The indexing for the subtally results is r, theta, z.

The TallyContainer also provides a relative error function for convenience, which will squelch "divide-by-zero" errors given when a tally bin encountered no particles, setting the relative error to 'inf':

```
>>> rxntal.total.re
array([[[ 0.01526919]]])
```

If energy binning was used, the tally multiplier result will also have energy-binned tallies, indexed by energy-bin first, then r, theta, z:

```
>>> rxntal.binned.mean.shape
(200, 1, 1, 1)
```

The tally results can be sliced and printed alongside the lower and upper energy groups:

```
>>> lower = rxntal.bin_bounds.get_lower_bounds()
>>> upper = rxntal.bin_bounds.get_upper_bounds()
>>> for (l, u, mean, rel) in zip(lower, upper,
...         rxntal.binned.mean[:,0,0,0], rxntal.binned.re[:,0,0,0]):
...     print "%.4e to %.4e: %10.3g | %.2f" % (l, u, mean, rel)
...
1.9640e+07 to 2.0000e+07:          0 | inf
    [snip]
1.5000e-01 to 1.8400e-01:   1.16e-06 | 0.04
1.2500e-01 to 1.5000e-01:   1.82e-06 | 0.03
1.0000e-01 to 1.2500e-01:   3.64e-06 | 0.02
7.0000e-02 to 1.0000e-01:   9.33e-06 | 0.02
5.0000e-02 to 7.0000e-02:   1.09e-05 | 0.02
4.0000e-02 to 5.0000e-02:   7.33e-06 | 0.02
3.0000e-02 to 4.0000e-02:   8.19e-06 | 0.02
2.1000e-02 to 3.0000e-02:   7.66e-06 | 0.02
1.4500e-02 to 2.1000e-02:    5.2e-06 | 0.02
1.0000e-02 to 1.4500e-02:   2.99e-06 | 0.02
5.0000e-03 to 1.0000e-02:   2.41e-06 | 0.02
2.0000e-03 to 5.0000e-03:   7.89e-07 | 0.03
5.0000e-04 to 2.0000e-03:   1.32e-07 | 0.06
1.0000e-05 to 5.0000e-04:   9.89e-09 | 0.17
```

CHAPTER

SIXTEEN

OMNIBUS.CONVERTERS PACKAGE

The Omnibus converters package allows results from other code systems to be read into Omnibus postprocessing containers for easy data analysis. This enables straightforward integration of other common nuclear engineering codes with powerful tools such as the matplotlib plotting library and the pandas data analysis library.

16.1 omnibus.converters.mctal module

Read MCNP mctal files into Omnibus postprocessing data structures.

Caution: MCNP is unable to represent complex user input (e.g. cell unions and multiplier selections) in the mctal file. Sometimes zeroes in the various "meshes" returned by this reader represent these complicated expressions. The MCNP input (or output file) will be needed to properly interpret these instances.

Note: When reading multigroup data, this reader sets an artificial small value for the lower energy of the lowest bin (which is implicitly created by MCNP). For correctness, it should be replaced with the problem's lower cutoff energy for the tallied particle type.

class `omnibus.converters.mctal.`**`Kcode`**`(`**args*, ***kwargs*`)`

Bases: `omnibus.postprocess.field.Field`

Eigenvalue solution properties.

Attributes

`dims`	Dimension names for each axis.
`keff`	Average col/abs/trk-len keff
`keff_fom`	Average col/abs/trk-len keff figure of merit
`keff_stdev`	Average col/abs/trk-len keff standard deviation
`location`	Get a description of where this field has been sliced from.
`num_active_cycles`	
`num_cycles`	
`num_histories_per_cycle`	Number of histories used in this cycle
`shape`	Get the shape of the data stored in this field..

Methods

`as_dataframe`()	Convert the field to a Pandas dataframe for easier processing.
`axis`(dim)	Get the axis index corresponding to the given name.
`axis_index`(dim)	Get the axis index corresponding to the given dimension name.
`reorder`(newdims)	Reorder the indices of the data by dimension name.
`xs`(**kwargs)	Extract a hyperslab view of the tally field data.
`xs_by_index`(**kwargs)	Extract a hyperslab view of the tally field data.

keff
Average col/abs/trk-len keff

keff_fom
Average col/abs/trk-len keff figure of merit

keff_stdev
Average col/abs/trk-len keff standard deviation

num_histories_per_cycle
Number of histories used in this cycle

class `omnibus.converters.mctal.`**McnpAxis**(**args*, ***kwargs*)
Bases: `omnibus.postprocess.field.LabelAxis`

Attributes for one 'dimension' of a tally.

Total and cumulative values are counted as part of the total length.

Attributes

mnemonic: string	Short name of the MCNP variable index.

Methods

`describe_index`(i)	
`index`(value)	Find the index of a value on this mesh.

index(*value*)
Find the index of a value on this mesh.

shape
The actual shape of this axis' mesh.

class `omnibus.converters.mctal.`**McnpCellTallies**(**args*, ***kwargs*)
Bases: `omnibus.postprocess.celltally.CellTallies`

Collection of all cell tallies present in a file.

Attributes

kcode :	Eigenvalue solution data

Methods

`from_group`(root)	Build results for all tallies in a "tallies" group.
`iteritems`()	
`iterkeys`()	
`itervalues`()	

class `omnibus.converters.mctal.`**`McnpCellTallyResult`**(**args*, ***kwargs*)

Bases: `omnibus.postprocess.celltally.CellTallyResult`

Data corresponding to a single cell tally.

This corresponds to all the results from a single fNN specification in MCNP.

Attributes

name	(int) Tally number
pt	(str) Particle type: n p e
taltype	(str) Tally type: none point ring fip fir fic
axes	(list) Dimensions on the tally data
tfc :	Tally fluctuation info.

Methods

`at_step`(step)	Return a view of the tally data at a single depletion time step.
`from_group`(group, shared)	Build results for a single CELL tally from an HDF5 file.
`from_group_v1`(group, *args, **kwargs)	Backward compatibility: build tallies from older HDF5 files.
`from_group_v2`(group, shared)	Backward compatibility: build tallies from older HDF5 files.
`from_group_v3`(group, shared)	Build results for a single CELL tally from an HDF5 file.
`from_group_v4`(group, shared)	Build results for a single CELL tally from an HDF5 file.
`summarize`([file])	Print a summary of the energy-integrated results for this tally.

class `omnibus.converters.mctal.`**`McnpTallyField`**(**args*, ***kwargs*)

Bases: `omnibus.postprocess.field.Field`

Multi-dimensional field for storing and accessing bulk tally data.

This provides accessors for retrieving the originally stored mean and variance, as well as calculation methods for returning relative error. Additionally, in the case of multi-dimensional data, it allows advanced slicing of the data.

Attributes

mean	(array (or float)) The mean values for this tally.
re	(array (or float)) Relative errors saved from MCNP.

Methods

`as_dataframe`()	Convert the field to a Pandas dataframe for easier processing.

Continued on next page

Table 16.6 – continued from previous page

`axis`(dim)	Get the axis index corresponding to the given name.
`axis_index`(dim)	Get the axis index corresponding to the given dimension name.
`reorder`(newdims)	Reorder the indices of the data by dimension name.
`xs`(**kwargs)	Extract a hyperslab view of the tally field data.
`xs_by_index`(**kwargs)	Extract a hyperslab view of the tally field data.

`bin_bounds`
If this is an energy-binned tally, return the bin bounds.

If not energy-binned (or a slice on the energy axis has already been taken), this will raise a `KeyError`.

`cells`
Get an integer array of MCNP cells.

`var`
Calculate variance from the relative error.

This will return an array if the stored mean and variance are arrays.

$$\text{re} = \frac{\sqrt{\text{var}}}{\text{mean}}$$

class `omnibus.converters.mctal.`**`Reader`**(*filepath*)
Bases: `omnibus.textfile.TextFile`

Helper class to read mctal file into tally data structure.

Attributes

`closed`	
`eof`	Return whether we're at the end of the file

Methods

`close`()	
`load`()	Read and parse the file, setting `tallies`.
`next`()	Get the next line of the file output, using a one-line buffer.
`read_values`(num)	Read `num` space-separated values from the file.

`load`()
Read and parse the file, setting `tallies`.

`read_values`(*num*)
Read `num` space-separated values from the file.

`tallies` **= None**
Tally data

`version` **= None**
Version number for switching on format

class `omnibus.converters.mctal.`**`Tfc`**(**args*, ***kwargs*)
Bases: `omnibus.container.Container`

Tally fluctuation properties.

Attributes

index :	Index into tally bin we're reporting
data	(dtype=Tfc.dtype) Record field of data

16.2 omnibus.converters.meshtal module

Read MCNP meshtal files into Omnibus postprocessing data structures.

class `omnibus.converters.meshtal.`**`McnpMeshTallies`**(**args*, ***kwargs*)
Bases: `omnibus.postprocess.meshtally.MeshTallies`

Collection of all MCNP mesh tallies present in a file.

Methods

`from_group`(root)	Build results for all tallies in a "tallies" group.
`iteritems`()	
`iterkeys`()	
`itervalues`()	

class `omnibus.converters.meshtal.`**`McnpMeshTallyResult`**(**args*, ***kwargs*)
Bases: `omnibus.postprocess.meshtally.MeshTallyResult`

Data corresponding to a single user tally.

Attributes

name	(int) Tally number
pt	(str) Particle type: n p e

Methods

`at_step`(step)	Return a view of the tally data at a single depletion time step.
`from_group`(group, shared)	Build results for a single MESH tally from an HDF5 file.
`from_group_v1`(group, shared)	Backward compatibility: build tallies from older HDF5 files.
`from_group_v2`(group, shared)	
`from_group_v3`(group, shared)	
`from_group_v4`(group, shared)	Build results for a single MESH tally from an HDF5 file.

`energy_bins`
Return energy bin boundaries (backwards compatibility)

16.3 omnibus.converters.monaco module

Read Monaco tallies into Omnibus postprocessing data structures.

Warning: Currently this class is only tested for multigroup cell tallies.

Note: When reading multigroup data, this reader sets an arbitrarily small value for the lower energy of the lowest bin (which is implicitly created by Monaco). For correctness, it should be replaced with the problem's lower cutoff energy for the tallied particle type.

class `omnibus.converters.monaco.`**`MonacoCellTallyResult`**`(*args, **kwargs)`

Bases: `omnibus.postprocess.tally.TallyResult`

Results for a single cell tally in Monaco.

Attributes

limits	(dict) Key->value map of region/unit/media on the input tally.
volume	(float) Volume of the tally.
multiplier	(float) Scalar multiplier used for the tally.

Methods

`at_step`(step)	Return a view of the tally data at a single depletion time step.
`from_group`(group, shared[, reorder])	Build results for a single tally from an HDF5 group.

16.4 omnibus.converters.opus module

16.5 omnibus.converters.triton module

Convert Triton number density output to same format as Omnibus depletion concentration field.

Note: This converter relies on an external tool to convert a Triton Fortran intermediate file to a `csv` file. It additionally requires cell volumes and the total heavy metal mass to scale the results correctly.

16.6 omnibus.converters.vesta module

16.7 Module contents

Convert third-party results into Omnibus postprocessing formats.

CHAPTER

SEVENTEEN

OMNIBUS.POSTPROCESS PACKAGE

17.1 Submodules

17.2 omnibus.postprocess.celltally module

class `omnibus.postprocess.celltally.`**`CellTallies`**(**args*, ***kwargs*)

Bases: `omnibus.postprocess.tally.Tallies`

Collection of all cell tallies present in a file.

Methods

`from_group`(root)	Build results for all tallies in a "tallies" group.
`iteritems`()	
`iterkeys`()	
`itervalues`()	

class `omnibus.postprocess.celltally.`**`CellTallyMultiplierResult`**(**args*, ***kwargs*)

Bases: `omnibus.postprocess.tally.TallyMultiplierResult`

Cell union tally results for a single tally multiplier.

These store all the cell unions specified by the user for this multiplier. A "multiplier" can be a reaction or a response.

Energy-binned tallies will have the shape *(num_groups, num_unions)*.

Attributes

unions	(list of strings) Cell union labels corresponding to each tally entry.

`num_unions`

The number of cell unions in this tally.

class `omnibus.postprocess.celltally.`**`CellTallyResult`**(**args*, ***kwargs*)

Bases: `omnibus.postprocess.tally.TallyResult`

Results for a single CELL tally, with all its multipliers.

This corresponds to all the results from a single [TALLY][CELL] block in an Omnibus input.

Attributes

unions	(list of strings) Cell union labels corresponding to each tally entry.
volumes	(array) Volumes used when the cell tally results were normalized. These may be given by the user, calculated via MCNP or the like, or unity if unknown. Variances and means in the tallies have *already* been normalized by these volumes.

Methods

summarize :	Print a summary of the energy-integrated results for this tally.

classmethod **`from_group`**(*group*, *shared*)
Build results for a single CELL tally from an HDF5 file.

This creates the union labels from cell lengths and *ids*. We use the shared data to convert cellids to cell labels.

classmethod **`from_group_v1`**(*group*, **args*, ***kwargs*)
Backward compatibility: build tallies from older HDF5 files.

classmethod **`from_group_v2`**(*group*, *shared*)
Backward compatibility: build tallies from older HDF5 files.

This creates the union labels from cell lengths and labels.

classmethod **`from_group_v3`**(*group*, *shared*)
Build results for a single CELL tally from an HDF5 file.

classmethod **`from_group_v4`**(*group*, *shared*)
Build results for a single CELL tally from an HDF5 file.

This creates the union labels from cell lengths and *ids*. We use the shared data to convert cellids to cell labels.

`num_unions`
The number of cell unions in this tally.

`summarize`(*file=None*)
Print a summary of the energy-integrated results for this tally.

Parameters **file** : file-like object

The destination for the output stream. Default is `sys.stdout`.

class `omnibus.postprocess.celltally.`**`SharedCellTallyData`**(**args*, ***kwargs*)
Bases: `omnibus.postprocess.tally.SharedTallyData`

Data shared by all tallies in a file.

This is mostly used in construction to pass problem metadata etc. between tallies.

Attributes

cell_labels: array	Labels of all cells in the problemzz

Methods

from_group(root)

class omnibus.postprocess.celltally.**SharedCellTallyMultiplierData**(**args, **kwargs*)

Bases: omnibus.postprocess.celltally.SharedCellTallyData

Data shared by a single cell tally.

Attributes

cell_unions: array	Labels of all cells in the problemzz

Methods

from_group(root)

class omnibus.postprocess.celltally.**SingleTallyContainer**

Bases: tuple

Attributes

binned	Alias for field number 1
label	Alias for field number 0
total	Alias for field number 2

Methods

count(...)	
index((value, [start, ...)	Raises ValueError if the value is not present.

binned
Alias for field number 1

label
Alias for field number 0

total
Alias for field number 2

omnibus.postprocess.celltally.**main**(*argv=None*)
Convert HDF5 cell tallies to CSV files.

17.3 omnibus.postprocess.collisions module

class omnibus.postprocess.collisions.**CollisionDiagnostic**(**args, **kwargs*)

Bases: omnibus.container.Container

Collision diagnostic results for a problem.

The collision diagnostic calculates the average number of collisions per history, and average weight loss per history, in each material, in each nuclide, for each reaction.

Attributes

data	(pd.DataFrame or np.array) Record array of materials, zaids, MT numbers, and collision/weight loss per history.
metadata	(dict) Metadata associated with program run.
num_histories	(float) Number of active particle histories sampled.
num_collisions	(float) Total number of collisions during active histories.

Methods

`extract`(**kwargs)	Extract a cross section view for a single key (matid/zaid/mt).
`from_h5_group`(group[, name, version])	
`integrate`(key)	Integrate collision results over all keys but the given one.
`reindex_from_materials`(mixtures)	Change index from matid to material name.
`summarize`([threshold, file])	Print a summary of important collisions in this tally.

extract(***kwargs*)
Extract a cross section view for a single key (matid/zaid/mt).

```
>>> df.extract(mt=2)
```

integrate(*key*)
Integrate collision results over all keys but the given one.

material_indexing
Return whether we're indexed by 'matid' or 'mat'.

reindex_from_materials(*mixtures*)
Change index from matid to material name.

This permanently changes the key name; you'll have to reload the data if you want matids back.

```
>>> xmldata = OmnibusOutput.from_xml('output.xml')
>>> mixtures = xmldata.db['PHYSICS']['mixtures']
>>> coldat.reindex_from_materials(mixtures)
```

summarize(*threshold=0.01*, *file=None*)
Print a summary of important collisions in this tally.

This integrates over all materials, printing a list of reactions with the most collisions per history (over the given threshold value).

Parameters **threshold** : float

Fraction of total collisions per history at which we truncate output of most frequent nuclides per collisino

file : file-like object

The destination for the output stream. Default is `sys.stdout`.

`omnibus.postprocess.collisions.`**`main`**`(`*argv=None*`)`

Print a summary of the collision diagnostic.

You can optionally pass a dictionary of mixture names to change the database's matids to mixture names.

17.4 omnibus.postprocess.compositions module

class `omnibus.postprocess.compositions.`**`Composition`**`(`**args*, ***kwargs*`)`

Bases: `omnibus.container.Container`

A single composition.

Attributes

name	(string) Composition name.
components	(dict) Map of ZAID -> number densities (atoms / b-cm)

Methods

`from_group`(root)	Build results for all compositions in the file.

classmethod **`from_group`**`(`*root*`)`

Build results for all compositions in the file.

class `omnibus.postprocess.compositions.`**`Comps`**`(`**args*, ***kwargs*`)`

Bases: `omnibus.container.Container`

All composition data from an HDF5 file

Attributes

compositions	(list of Composition) All compositions from the file
nuclides	(list of Nuclide) All nuclides from the file
metadata	(dict) Problem metadata.

Methods

`from_group`(root)	Build results for all compositions in the file.

classmethod **`from_group`**`(`*root*`)`

Build results for all compositions in the file.

class `omnibus.postprocess.compositions.`**`Nuclide`**`(`**args*, ***kwargs*`)`

Bases: `omnibus.container.Container`

A single nuclide property.

Attributes

name	(string) Nuclide name.
mass	(float) Atomic mass (g/mol)
zaid	(int) SCALE nuclide identifier.

Methods

`from_group`(root[, zaid])	

`omnibus.postprocess.compositions.`**`load`**`(`*handle*`)`
Read composition data from an HDF5 file.

17.5 omnibus.postprocess.cyltally module

class `omnibus.postprocess.cyltally.`**`CylMesh`**`(`**args*, ***kwargs*`)`
Bases: `omnibus.container.Container`

Container for a cylindrical mesh.

The Cylindrical mesh is indexed with [r, z, theta].

Attributes

r	(array) Radial mesh points.
z	(array) Z-axis mesh points.
t	(array) Azimuthal (theta) mesh points.
translation	(array) Length-3 translation for the cylindrical mesh.
rotation	(array) 3x3 rotation matrix for the cylindrical mesh.

`cell_shape`
The proper shape for cell-centered data associated with the mesh.

returns: (r, z, theta).

class `omnibus.postprocess.cyltally.`**`CylTallies`**`(`**args*, ***kwargs*`)`
Bases: `omnibus.postprocess.tally.Tallies`

Collection of all cylindrical tallies present in a file.

Methods

`from_group`(root)	Build results for all tallies in a “tallies” group.
`iteritems`()	
`iterkeys`()	
`itervalues`()	

class `omnibus.postprocess.cyltally.`**`CylTallyMultiplierResult`**`(`**args*, ***kwargs*`)`
Bases: `omnibus.postprocess.tally.TallyMultiplierResult`

Cylindrical mesh tally results for a single tally multiplier.

The cylindrical multiplier tally result reshapes the data to conform with the cylindrical mesh: energy-integrated tally results will have the indexing [r, z, theta], and energy-binned tallies are accessed like [e, r, z, theta].

A "multiplier" can be a reaction or a response.

Attributes

mesh	(`CylMesh`) The mesh used in this tally

class `omnibus.postprocess.cyltally.`**`CylTallyResult`**(**args*, ***kwargs*)

Bases: `omnibus.postprocess.tally.TallyResult`

Results for a single CYLMESH tally, with all its multipliers.

This corresponds to all the results from a single [TALLY][CYLMESH] block in an Omnibus input.

Attributes

mesh	(`CylMesh`) The mesh used in this tally

Methods

write_to_silo :	Write all the embedded tallies to a Silo file.

classmethod **`from_group`**(*group*, *shared*)

Build results for a single CYLMESH tally from an HDF5 file.

The version keyword is the Exnihilo revision number, used for backward compatibility.

classmethod **`from_group_v1`**(*group*, *shared*)

Backward compatibility: build tallies from older HDF5 files.

classmethod **`from_group_v4`**(*group*, *shared*)

Build results for a single CYLMESH tally from an HDF5 file.

The version keyword is the Exnihilo revision number, used for backward compatibility.

17.6 omnibus.postprocess.depletion module

class `omnibus.postprocess.depletion.`**`Concentrations`**(**args*, ***kwargs*)

Bases: `omnibus.postprocess.depletion.DepletionField`

Concentrations of every nuclide in each cell at each step.

Attributes

data	(array of doubles) Array of all concentrations. [step][cell][zaid]

Methods

to_num_dens :	Convert concentrations from mol -> atoms/barn-cm.

class `omnibus.postprocess.depletion.`**`CrossSections`**`(*args, **kwargs)`
Bases: `omnibus.postprocess.depletion.DepletionField`

Collapsed ORGEN one-group cross sections.

These are provided for every ORIGEN zaid and mt in each cell at each step.

Attributes

`xs`(**kwargs)	Extract a hyperslab view of the tally field data.

Methods

write_to_csv:	Write the cross sections to a CSV file.

`as_dataframe``()`
Convert the cross sections to a dataframe for easier processing.

The zaids and MTs are expanded into separate axes.

class `omnibus.postprocess.depletion.`**`DepletionField`**`(*args, **kwargs)`
Bases: `omnibus.postprocess.field.Field`

Store multi-D data with axes and labels.

Unlike the tally field, this doesn't have variances, and they aren't necessarily statistically calculated quantities.

Attributes

`dims`	Dimension names for each axis.
`location`	Get a description of where this field has been sliced from.
`shape`	Get the shape of the data stored in this field..

Methods

`as_dataframe`()	Convert the field to a Pandas dataframe for easier processing.
`axis`(dim)	Get the axis index corresponding to the given name.
`axis_index`(dim)	Get the axis index corresponding to the given dimension name.
`from_group`(group, key)	
`reorder`(newdims)	Reorder the indices of the data by dimension name.
`xs`(**kwargs)	Extract a hyperslab view of the tally field data.
`xs_by_index`(**kwargs)	Extract a hyperslab view of the tally field data.

class `omnibus.postprocess.depletion.`**`DepletionTally`**`(*args, **kwargs)`
Bases: `omnibus.container.Container`

Results from a depletion run.

Attributes

num_dens	(array of doubles) Array of number densities. [step][cell][zaid]
concentrations	(array of doubles) Array of concentrations. [step][cell][zaid]
volumes	(array of doubles) Array of volumes for all depletion regions. [cell]
power	(list of doubles) Array of power in all regions if depleting by power. [cell]
flux	(list of doubles) Array of flux in all regions if depleting by flux. [cell]
hm_fraction	(scalar double) The fraction of heavy metal mass in the system (for normalization)
metadata	(array) The metadata from the problem run.
simtime	(array of doubles) Array of running simulation time of the problem [step]
burnup	(array of doubles) Array of the running burnup of the problem (MWd/MTU) [step]
hm_mass	(scalar double) The total heavy metal mass in the system at beginning of calculation.
burn_time	(array of doubles) Array of the burn length per step size (days). [step]
decay_time	(array of doubles) Array of the decay length per time step (days). [step]
totpower	(array of doubles) Array of total power in the system for each time step. [step]
xs	(array of doubles) Array of collapsed 1-grp cross sections from ORIGEN. [step][cell][zaid-mt]

Methods

at_step :	Get a slice of the DepletionTally for a given time step number.
summarize :	Print a summary of the burn data for every time step.

`at_step`(*step*)

Return a view of the tally data at a single depletion time step.

Returns DepletionTally (or subclass): tally result evaluated at the given time

step.

`convert_to_time`()

Return a new depletion tally with only physical time steps.

This effectively removes data for “predictor” steps if present, allowing easier comparison with other depletion codes. It also replaces the “step” index with a “time” index.

It does not modify the original depletion tally.

Returns New depletion tally indexed by time.

classmethod `from_group`(*root*)

Build results for all depletion data.

classmethod `from_group_v1`(*root*)

Update version 1 depletion data to version 2.

This requires:

- Modifying dimension names to change nuclide to zaid
- Combining the ZAID and MT fields

classmethod `from_group_v2`(*root*)

Update version 2 depletion data to version 3.

This requires:

- Convert concentrations to number densities

- Make cell labels in region data into root field

summarize(*file=None*)

Print a summary of the depletion burn data.

Parameters file : file-like object

The destination for the output stream. Default is `sys.stdout`.

class `omnibus.postprocess.depletion.`**`InputField`**(**args*, ***kwargs*)

Bases: `omnibus.postprocess.depletion.DepletionField`

Store multi-D input data.

Attributes

`dims`	Dimension names for each axis.
`location`	Get a description of where this field has been sliced from.
`shape`	Get the shape of the data stored in this field..

Methods

`as_dataframe`()	Convert the field to a Pandas dataframe for easier processing.
`axis`(dim)	Get the axis index corresponding to the given name.
`axis_index`(dim)	Get the axis index corresponding to the given dimension name.
`from_group`(group, key)	
`reorder`(newdims)	Reorder the indices of the data by dimension name.
`xs`(**kwargs)	Extract a hyperslab view of the tally field data.
`xs_by_index`(**kwargs)	Extract a hyperslab view of the tally field data.

class `omnibus.postprocess.depletion.`**`NumberDensities`**(**args*, ***kwargs*)

Bases: `omnibus.postprocess.depletion.DepletionField`

Number density of every nuclide in each cell at each step.

Attributes

data	(array of doubles) Array of all number densities. [step][cell][zaid]

Methods

write_to_csv :	Write the number densities to a CSV file.

class `omnibus.postprocess.depletion.`**`RegionData`**(**args*, ***kwargs*)

Bases: `omnibus.postprocess.depletion.DepletionField`

Data about every depletion region at each step.

Attributes

`dims`	Dimension names for each axis.
`location`	Get a description of where this field has been sliced from.
`shape`	Get the shape of the data stored in this field..

Methods

`as_dataframe`()	Convert the field to a Pandas dataframe for easier processing.
`axis`(dim)	Get the axis index corresponding to the given name.
`axis_index`(dim)	Get the axis index corresponding to the given dimension name.
`from_group`(group, key)	
`reorder`(newdims)	Reorder the indices of the data by dimension name.
`xs`(**kwargs)	Extract a hyperslab view of the tally field data.
`xs_by_index`(**kwargs)	Extract a hyperslab view of the tally field data.

`omnibus.postprocess.depletion.`**`main`**(*argv=None*)

Write HDF5 depletion results to CSV files.

17.7 omnibus.postprocess.field module

This module provides a wrapper to Numpy arrays that stores multidimensional data, allowing it to be sliced, reordered, and easily accessed.

class `omnibus.postprocess.field.`**`DataAxis`**(**args*, ***kwargs*)

Bases: `omnibus.container.Container`

Metadata about an axis of a field.

The mesh must be sorted in ascending order for the index method to work.

Attributes

`centers`	Obtain mesh values that match the centering of the data.
`shape`	The actual shape of this axis' mesh.

Methods

`describe_index`(i)	
`index`(value)	Find the index of a value on this mesh.

`centers`

Obtain mesh values that match the centering of the data.

`index`(*value*)

Find the index of a value on this mesh.

`shape`

The actual shape of this axis' mesh.

class `omnibus.postprocess.field.`**`Field`**(**args*, ***kwargs*)

Bases: `omnibus.container.Container`

Multi-dimensional field for storing and accessing multi-D bulk data.

It enables advanced slicing of the data in this field. It should be subclassed to be used, and the `_fields` attribute should be set to the relevant fields.

Attributes

axes	(list of DataAxis) Attributes about each dimension/axis of the data.
hyper-slice	(list of (axis, index) tuples.) If this is a view into larger tally data, the hyperslice is the list of axes and values at which those dimensions are being evaluated.

Methods

`as_dataframe`()	Convert the field to a Pandas dataframe for easier processing.
`axis`(dim)	Get the axis index corresponding to the given name.
`axis_index`(dim)	Get the axis index corresponding to the given dimension name.
`reorder`(newdims)	Reorder the indices of the data by dimension name.
`xs`(**kwargs)	Extract a hyperslab view of the tally field data.
`xs_by_index`(**kwargs)	Extract a hyperslab view of the tally field data.

`as_dataframe`()

Convert the field to a Pandas dataframe for easier processing.

Fields are converted to series in the dataframe, and the axes are converted to a multi-index.

Note: It may be necessary to run call `result.sortlevel(inplace=True)` on the result to perform some Pandas operation. This is because some axes may not be lexicographically ordered.

`axis`(*dim*)

Get the axis index corresponding to the given name.

Examples

This enables easier access to meshes and the like.

```
>>> tal.axis('x').mesh
```

`axis_index`(*dim*)

Get the axis index corresponding to the given dimension name.

Examples

To collapse multi-dimensional data using the `numpy.sum` function, you can determine the appropriate axis index using this method:

```
>>> tal.total.sum(axis=tal.axis_index("x"))
```

`dims`

Dimension names for each axis.

location
Get a description of where this field has been sliced from.

It will return an empty string if this is the full, original field.

reorder(*newdims*)
Reorder the indices of the data by dimension name.

Parameters **newdims** : list of strings

New dimension names.

Examples

This allows conversion of data stored as ZYXE to XYZE, etc. The input argument is a list of dimensions that should end up in the relative order given:

```
>>> newvals = tal.reorder(("x","y"))
```

shape
Get the shape of the data stored in this field..

xs(***kwargs*)
Extract a hyperslab view of the tally field data.

Input arguments are values on each axis, similar to the pandas "xs" function.

xs_by_index(***kwargs*)
Extract a hyperslab view of the tally field data.

Input arguments must be integer indices, not values.

class omnibus.postprocess.field.**LabelAxis**(**args*, ***kwargs*)
Bases: omnibus.container.Container

Metadata about an axis of a field that acts like labels.

This is currently used for tally multipliers and unions.

Attributes

centers
shape

Methods

describe_index(i)
index(value)

omnibus.postprocess.field.**build_axes**(*group*, *dimnames*, *shape*)
From an HDF5 group or dict, build a list of axes.

17.8 omnibus.postprocess.manager module

class `omnibus.postprocess.manager.`**`JsonWriter`**

Bases: `object`

Write a nicer version of the XML output.

This writes a JSON version of the XML data entries.

Attributes

`outputs`

Methods

`__call__`(manager)
`write_rst`(w)

class `omnibus.postprocess.manager.`**`Manager`**(*db*, *filename*)

Bases: `object`

Manage output from an Omnibus run.

Attributes

db	(dict) Dictionary of parsed XML output.
dirname	(string) Directory enclosing the output.
basename	(string) Name of output file without extensions.

Methods

`finalize`()	Call to write an RST file after postprocesing.
`from_xml`(xml_output_path)	
`get_timing`(block, timer)	Return the node-zero time for the given timer in the given block.
`process`(p)	Handle postprocessing.

`finalize`()

Call to write an RST file after postprocesing.

`get_timing`(*block*, *timer*)

Return the node-zero time for the given timer in the given block.

`mixtures`

Return mixtures if available, or None if not.

`process`(*p*)

Handle postprocessing.

`shift_db`

Return the SHIFT execution block database.

If depletion is enabled, this only returns the *last* shift DB.

17.9 omnibus.postprocess.meshtally module

class `omnibus.postprocess.meshtally.`**`Mesh`**`(`**args*, ***kwargs*`)`
Bases: `omnibus.container.Container`

Container for a cartesian mesh.

The mesh is indexed with [x, y, z].

Attributes

x	(array) X-axis mesh points.
y	(array) Y-axis mesh points.
z	(array) Z-axis mesh points.

Methods

`from_group`(group)	

`cell_shape`
Return the proper shape for cell-centered data so it can be accessed with [x, y, z].

class `omnibus.postprocess.meshtally.`**`MeshTallies`**`(`**args*, ***kwargs*`)`
Bases: `omnibus.postprocess.tally.Tallies`

Collection of all mesh tallies present in a file.

Methods

`from_group`(root)	Build results for all tallies in a "tallies" group.
`iteritems`()	
`iterkeys`()	
`itervalues`()	

class `omnibus.postprocess.meshtally.`**`MeshTallyMultiplierResult`**`(`**args*, ***kwargs*`)`
Bases: `omnibus.postprocess.tally.TallyMultiplierResult`

Mesh tally results for a single tally multiplier.

The multiplier tally result reshapes the data to conform with the mesh: energy-integrated tally results will have the indexing [x, y, z], and energy-binned tallies are accessed like [bin, x, y, z].

A "multiplier" can be a reaction or a response.

Attributes

mesh	(`Mesh`) The mesh used in this tally

class `omnibus.postprocess.meshtally.`**`MeshTallyResult`**(**args*, ***kwargs*)

Bases: `omnibus.postprocess.tally.TallyResult`

Results for a single MESH tally, with all its multipliers.

This corresponds to all the results from a single [TALLY][MESH] block in an Omnibus input.

Attributes

mesh	(`Mesh`) The mesh used in this tally

Methods

`at_step`(step)	Return a view of the tally data at a single depletion time step.
`from_group`(group, shared)	Build results for a single MESH tally from an HDF5 file.
`from_group_v1`(group, shared)	Backward compatibility: build tallies from older HDF5 files.
`from_group_v2`(group, shared)	
`from_group_v3`(group, shared)	
`from_group_v4`(group, shared)	Build results for a single MESH tally from an HDF5 file.

classmethod **`from_group`**(*group*, *shared*)

Build results for a single MESH tally from an HDF5 file.

classmethod **`from_group_v1`**(*group*, *shared*)

Backward compatibility: build tallies from older HDF5 files.

classmethod **`from_group_v4`**(*group*, *shared*)

Build results for a single MESH tally from an HDF5 file.

17.10 omnibus.postprocess.rst module

class `omnibus.postprocess.rst.`**`RstBlockType`**

Bases: `type`

Metaclass used for easily generating RST writers for each block.

Methods

`__call__`(...) <==> x(...)	
`mro`(() -> list)	return a type's method resolution order

17.11 omnibus.postprocess.sensitivity module

17.12 omnibus.postprocess.tally module

This module contains base classes for tally postprocessing.

class `omnibus.postprocess.tally.`**`BinAxis`**(**args*, ***kwargs*)

Bases: `omnibus.container.Container`

Metadata about an axis of a tally for multigroup data.

Attributes

`shape`

Methods

`describe_index`(i)
`index`(value)

class `omnibus.postprocess.tally.`**`BinBounds`**(**args*, ***kwargs*)

Bases: `omnibus.container.Container`

Bin boundaries when tallying over energy or particle.

This includes energy bounds for all particle types. If no boundaries are present, `bool(bb)` will return `False`.

Examples

Since Omnibus tallies contain both neutron and photon data, one common requirement is to view tally data from a single particle. For this purpose, the `BinBounds` class contains a `slice_for_particle` method, which returns a Python `slice` object corresponding to a single particle type. For example, you can take an energy-dependent tally in a single cell and plot only the neutron fluxes:

```
>>> bin_bounds = tally.bin_bounds
>>> slc = bin_bounds.slice_for_particle('n')
>>> plt.plot(bin_bounds.lower_bounds[slc], tally.mean[slc],
...          drawtype='steps-lower')
```

Attributes

n	(array or None) If the particle is being tallied, neutron bin boundaries in descending order. If not being tallied, this is None.
p	(array or None) If the particle is being tallied, photon bin boundaries in descending order. If not being tallied, this is None.

Methods

`describe_group`(g)	
`from_group`(group)	Create a `BinBounds` from an HDF5 group.
`labeled_groups`([particle])	Particle and relative group labels for each stored bin.
`num_groups`(particle)	Return number of groups for a given particle.
`slice_for_particle`(particle_type)	Return a slice object that will give the range of absolute group indices for a particle.

bounds
Returns the tuple of group bounds (neutron, photon)

classmethod from_group (*group*)
Create a BinBounds from an HDF5 group.

Returns BinBounds object if bin boundaries are present, else *None*.

labeled_groups (*particle=None*)
Particle and relative group labels for each stored bin.

This generator method yields a tuple (particle type, relative group) for all groups of particle type *particle* (or both if particle is None).

Parameters **particle** : int

Particle type (default None for both particles), one of *NEUTRON* or *PHOTON*.

lower_bounds
Get an array of lower energy bounds for all groups.

This is primarily useful when both photon and neutron groups are present, where the lower/upper bounds have a discontinuity.

num_groups (*particle*)
Return number of groups for a given particle.

num_total_groups
The number of groups summed over all particles.

slice_for_particle (*particle_type*)
Return a slice object that will give the range of absolute group indices for a particle.

Use this to extract subsets of collapsed multigroup data, or use the “start” and “stop” attributes of the slice object if you need those.

upper_bounds
Get an array of upper energy bounds for all groups.

This is primarily useful when both photon and neutron groups are present, where the lower/upper bounds have a discontinuity.

class omnibus.postprocess.tally.**DifferenceField** (**args*, ***kwargs*)
Bases: omnibus.postprocess.field.Field

Calculate the differences between two tallies.

This field stores the difference (relative error) between two tallies, and propagates the uncertainty between them.

The difference saved is:

$$\text{diff} = \frac{\text{mean}}{\text{mean}_{\text{ref}}} - 1$$

and the variance is:

$$\sigma^2_{\text{diff}} = \left[\frac{\text{mean}}{\text{mean}_{\text{ref}}}\right]^2 \left[\frac{\sigma^2}{\text{mean}^2} + \frac{\sigma^2_{\text{ref}}}{\text{mean}_{\text{ref}}}\right]$$

It accepts a threshold value for the reference data comparisons. Expected data points with a relative error greater than or equal to this value will be ignored.

Attributes

delta	(array) The relative difference between the two given tallies.
var	(array) The variances corresponding to the means.
axes	(list of DataAxis) Attributes about each dimension/axis of the data.
hyper-slice	(list of (axis, index) tuples.) If this is a view into larger tally data, the hyperslice is the list of axes and values at which those dimensions are being evaluated.

Methods

`as_dataframe`()	Convert the field to a Pandas dataframe for easier processing.
`axis`(dim)	Get the axis index corresponding to the given name.
`axis_index`(dim)	Get the axis index corresponding to the given dimension name.
`from_tallies`(reference, given[, re_threshold])	Build from two tally fields.
`reorder`(newdims)	Reorder the indices of the data by dimension name.
`xs`(**kwargs)	Extract a hyperslab view of the tally field data.
`xs_by_index`(**kwargs)	Extract a hyperslab view of the tally field data.

classmethod `from_tallies`(*reference*, *given*, *re_threshold=0.707*)
Build from two tally fields.

class `omnibus.postprocess.tally.`**`SharedTallyData`**(**args*, ***kwargs*)
Bases: `omnibus.container.Container`

Data shared by all tallies in a file.

This is mostly used in construction to pass problem metadata etc. between tallies. It may include data (such as tally_name) that are changed depending on what tally is currently being processed.

Methods

`from_group`(root)

class `omnibus.postprocess.tally.`**`Tallies`**(**args*, ***kwargs*)
Bases: `omnibus.container.Container`

Collection of all tallies present in a file.

Attributes

metadata :	Information about the run that created this tally data.

Methods

`from_group`(root)	Build results for all tallies in a “tallies” group.
`iteritems`()	
`iterkeys`()	
`itervalues`()	

classmethod **`from_group`**(*root*)
Build results for all tallies in a “tallies” group.

class `omnibus.postprocess.tally.`**`TallyField`**(**args*, ***kwargs*)
Bases: `omnibus.postprocess.field.Field`

Multi-dimensional field for storing and accessing bulk tally data.

This provides accessors for retrieving the originally stored mean and variance, as well as calculation methods for returning relative error. Additionally, in the case of multi-dimensional data, it allows advanced slicing of the data.

Attributes

mean	(array (or float)) The mean values for this tally.
var	(array (or float)) The variances corresponding to the means.
axes	(list of DataAxis) Attributes about each dimension/axis of the data.
hyper-slice	(list of (axis, index) tuples.) If this is a view into larger tally data, the hyperslice is the list of axes and values at which those dimensions are being evaluated.

Methods

`as_dataframe`()	Convert the field to a Pandas dataframe for easier processing.
`axis`(dim)	Get the axis index corresponding to the given name.
`axis_index`(dim)	Get the axis index corresponding to the given dimension name.
`from_group`(group, name)	
`reorder`(newdims)	Reorder the indices of the data by dimension name.
`without_last_energy_bin`()	Return a new tally field with the last energy bin removed.
`xs`(**kwargs)	Extract a hyperslab view of the tally field data.
`xs_by_index`(**kwargs)	Extract a hyperslab view of the tally field data.

`bin_bounds`
If this is an energy-binned tally, return the bin bounds.

If not energy-binned (or a slice on the energy axis has already been taken), this will raise a `KeyError`.

`re`
Calculate relative error, returning inf if mean is zero.

This will return an array if the stored mean and variance are arrays.

$$\mathrm{re} = \frac{\sqrt{\mathrm{var}}}{\mathrm{mean}}$$

`without_last_energy_bin`()
Return a new tally field with the last energy bin removed.

This is useful for getting Shift, Monaco, and MCNP tallies to match (because the latter two implicitly add a cutoff-to-lowest-energy bin).

If not energy-binned (or a slice on the energy axis has already been taken), this will raise a `KeyError`.

class `omnibus.postprocess.tally.`**`TallyMultiplierResult`**(**args*, ***kwargs*)
Bases: `omnibus.container.Container`

Tally results for a single tally multiplier.

This contains the final values of multigroup and total data.

Attributes

name	(string) Short label of this multiplier.
description	(string) Longer description of the multiplier.
bin_bounds	(`BinBounds`) Bin boundaries for multigroup tallies, or *None* for tallies without binning.
total	(`TallyField`) Energy- and particle-integrated means and variances for this multiplier.
binned	(`TallyField`) Energy-binned means and variances for this multiplier.

`has_energy_bins`
Whether this multiplier has energy-binned tallies.

class `omnibus.postprocess.tally.`**`TallyResult`**`(`**args*, ***kwargs*`)`
Bases: `omnibus.container.Container`

Set of results from a single tally.

This contains `TallyMultiplierResult` objects, which contain the data for each multiplier.

Attributes

`has_energy_bins`	Whether this multiplier has energy-binned tallies.

name	(string) Tally name.
descrip-tion	(string) User-specified description; defaults to value of *name*.
bin_bounds	(`BinBounds`) Bin boundaries for multigroup tallies. Evaluates as False if no boundaries are present.
tallies	(dict) Multiplier results for this tally, keyed by short name.
metadata	(dict) Execution metadata associated with this tally: problem ID, etc.

Methods

`at_step`(step)	Return a view of the tally data at a single depletion time step.
`from_group`(group, shared[, reorder])	Build results for a single tally from an HDF5 group.

`at_step``(`*step*`)`
Return a view of the tally data at a single depletion time step.

If depletion is disabled, this will raise an error.

Returns Tally (or subclass): tally result evaluated at the given time step.

classmethod **`from_group`**`(`*group*, *shared*, *reorder=None*`)`
Build results for a single tally from an HDF5 group.

It uses slices of the multi-tally data to create multiplier results.

`has_energy_bins`
Whether this multiplier has energy-binned tallies.

`multipliers`
Get a list of multipliers used in this tally.

`num_multipliers`
Number of multipliers.

`omnibus.postprocess.tally.`**`get_tally_version`**(*metadata*)
Return the tally compatibility version number.

This version number is only used internally and is not exposed to the user.

`omnibus.postprocess.tally.`**`load_tallies_from_h5`**(*handle*, *Tallies_cls*)
Load standard-format replicated HDF5 tallies from a file.

This prints out appropriate metadata while loading.

17.13 omnibus.postprocess.tally_plotter module

17.14 omnibus.postprocess.utils module

Common functions for postprocessing.

exception `omnibus.postprocess.utils.`**`NoDataException`**
Bases: `exceptions.Exception`

An exception class used by postprocessing.

This indicates that no suitable data is present in an output file, but that this is not an abnormal condition.

class `omnibus.postprocess.utils.`**`PostProcessOutput`**
Bases: `object`

Abstract base class for postprocess outputters.

A postprocessor output should raise a NoDataException in the constructor if inapplicable. It should have a "description" class attribute to provide a useful message if postprocessing fails. It should also have a "block" attribute for determining where in the RST output it should be called.

If postprocessing succeeds, the object will be retained so that it can provide more informative output to the user at the end of the run.

Attributes

`outputs`

block	
description	

Methods

`__call__`(manager)
`write_rst`(rstwriter)

`omnibus.postprocess.utils.`**`float_from_h5`**(*value*)
Extract a floating point value from an HDF5 file.

This is required because older HDF5 implementations wrote a length-1 array instead of a scalar.

`omnibus.postprocess.utils.`**`group_to_dict`**(*h5_root*)
Utility function for converting HDF5 files to dictionaries.

This is useful for creating unit tests that will run on systems without HDF5 present.

`omnibus.postprocess.utils.`**`h5dump`**(*path*)
Print an HDF5 file as a nicely formatted string.

`omnibus.postprocess.utils.`**`int_from_h5`**(*value*)
Extract an integer from an HDF5 file.

This is required because older HDF5 implementations wrote a length-1 array instead of a scalar.

`omnibus.postprocess.utils.`**`load_metadata`**(*md*)
Load metadata from the 'metadata' group of an HDF5 file.

The resulting dictionary contains version info,

`omnibus.postprocess.utils.`**`load_pyplot`**()
A function to lazily load the Python plotter.

`omnibus.postprocess.utils.`**`string_from_h5`**(*value*)
Extract a null-terminated string from an HDF5 field.

17.15 Module contents

The user-accessible package for analysis of Omnibus output.

This postprocessing package provides utility functions and interfaces for processing data from an Omnibus run.

Part IV

User Guide: Insilico

CHAPTER

EIGHTEEN

INTRODUCTION

The Insilico package provides a front-end (**neutronics**) and libraries for running reactor applications using CASL's VERA input specification. This specification is documented in VERAInExt/verain/docs/verain_UM.pdf. The VERAInExt repository also contains the VERA input pre-processor. This repository is available through CASL. An example VERA input for a 17×17 PWR assembly is as follows:

```
[CASEID]
  title 'CASL AMA Problem 3a - Single 17x17 Assembly - Public'

[STATE]
  power  0.0            ! %
  tinlet 620.33         ! F - 600K
  tfuel  600.0          ! K
  boron  1300           ! ppmB
  modden 0.743          ! g/cc
  sym qtr
  feedback off

[CORE]
  size 1                ! one assembly
  rated 17.67 0.6823    ! MW, Mlbs/hr
  apitch 21.5
  height 406.337

  core_shape
    1

  assm_map
    ASSY

  lower_plate ss  5.0 0.5   ! mat, thickness, vol frac
  upper_plate ss  7.6 0.5
  lower_ref  mod 20.0 1.0

  bc_rad reflecting

  mat he     0.0001786
  mat zirc   6.56 zirc4
  mat inc    8.19
  mat ss     8.0

[ASSEMBLY]
  title "Westinghouse 17x17"
  npin 17
  ppitch 1.26
```

```

fuel U31 10.257 94.5 / 3.1

cell 1     0.4096 0.418 0.475 / U31 he zirc
cell 3            0.561 0.602 / mod    zirc       ! guide/instrument tube
cell 4            0.418 0.475 /     he zirc       ! plenum
cell 5                  0.475 /        zirc       ! end plug

lattice FUEL
     3
     1 1
     1 1 1
     3 1 1 3
     1 1 1 1 1
     1 1 1 1 1 3
     3 1 1 3 1 1 1
     1 1 1 1 1 1 1 1
     1 1 1 1 1 1 1 1 1

lattice PLEN
     3
     4 4
     4 4 4
     3 4 4 3
     4 4 4 4 4
     4 4 4 4 4 3
     3 4 4 3 4 4 4
     4 4 4 4 4 4 4 4
     4 4 4 4 4 4 4 4 4

lattice PLUG
     3
     5 5
     5 5 5
     3 5 5 3
     5 5 5 5 5
     5 5 5 5 5 3
     3 5 5 3 5 5 5
     5 5 5 5 5 5 5 5
     5 5 5 5 5 5 5 5 5

axial ASSY  10.281
      PLUG  11.951
      FUEL 377.711
      PLEN 393.711
      PLUG 395.381

grid END inc  1017 3.866
grid MID zirc 875  3.810

grid_axial
    END  13.884
    MID  75.2
    MID 127.4
    MID 179.6
    MID 231.8
    MID 284.0
    MID 336.2
```

```
      END 388.2

  lower_nozzle  ss 6.053 6250.0  ! mat, height, mass (g)
  upper_nozzle  ss 8.827 6250.0  ! mat, height, mass (g)

[EDITS]
  axial_edit_bounds
      11.951
      15.817
      24.028
      32.239
      40.45
      48.662
      56.873
      65.084
      73.295
      77.105
      85.17
      93.235
      101.3
      109.365
      117.43
      125.495
      129.305
      137.37
      145.435
      153.5
      161.565
      169.63
      177.695
      181.505
      189.57
      197.635
      205.7
      213.765
      221.83
      229.895
      233.705
      241.77
      249.835
      257.9
      265.965
      274.03
      282.095
      285.905
      293.97
      302.035
      310.1
      318.165
      326.23
      334.295
      338.105
      346.0262
      353.9474
      361.8686
      369.7898
      377.711

```

```
[INSILICO]
    cell_homogenize true
    eq_set         spn_fv
    SPN_order      3
    Pn_order       1
    tolerance      1e-6
    dimension      3
    Pn_correction true

    mesh           2
    max_delta_z    1.27
    num_groups     23

    mat_library casl_comp_r2.sh5
    xs_library  lib252_hetbondoneabs-noabssigp

!   pin_partitioning true
    num_blocks_i  4
    num_blocks_j  4
    num_z_blocks  1
    num_sets      1

    silo_output  p3

    new_grp_bounds
       8.2085e+05
       1.1109e+05
       5.5308e+03
       1.8644e+02
       3.7612e+01
       3.5379e+01
       2.7697e+01
       2.1684e+01
       2.0397e+01
       1.5968e+01
       7.1500e+00
       6.7000e+00
       6.3000e+00
       1.0970e+00
       1.0450e+00
       9.5000e-01
       3.5000e-01
       2.0600e-01
       1.0700e-01
       5.8000e-02
       2.5000e-02
       1.0000e-02
       1.0000e-05
```

In order to execute this problem through any VERA-supported application (including Insilico), it must be processed into an XML file. The `VERAInExt` **react2xml.pl** script is used to convert the ASCII input file to code-readable XML:

```
$ perl react2xml.pl 3a.inp 3a.xml
```

The XML can be edited directly by the user (you **better** know what you're doing!) as this provides a method for power users to directly affect code control settings that are not normally exposed at the pre-process level.

The XML file can then be fed to any VERA executable. The VERA executable provided by Insilico is **neutronics**. It does a full neutronics simulation on the specified input (ie. no TH-coupling, etc.). There are VERA drivers that run coupled simulations that are available through CASL.

In order to run **neutronics** do the following:

```
$ mpirun -np 4 ./neutronics -i 3a.xml
```

The neutronics simulation performs the following basic steps:

1. build problem model
2. build problem mesh or geometry
3. load and process cross section
4. run transport
5. produce integrated pin-power and flux output

Insilico currently supports 3 transport options from the Exnihilo packages Denovo and Shift

- Denovo S_N (discrete ordinates)
- Denovo SP_N (simplified spherical harmonics)
- Shift Monte Carlo

The complete parameter list specification is available internally for power users.

Part V

Appendices

CHAPTER

NINETEEN

SHIFT ACCEPTANCE TEST DESCRIPTIONS

19.1 Leakage test

These tests compare the leakage out of a 23.7 cm thick spherical shell with an inner radius of 1.3 cm in a void for various coupled neutron-photon problems. An isotropic Cf-252 Watt spectrum neutron spherical source with radius 0.1 cm is located at the center of the shell. An energy-binned tally calculates the neutron and photon flux exiting the sphere in a 1 cm thick spherical shell void region located at distance of 99.9 cm from the center of the problem.

Shift is run in fixed-source mode simulating 1e7 particle histories using continuous-energy and multigroup physics for various nuclides. A Shift reference solution for all of these tests is used to compare that the flux in each bin for each test is within 4 standard deviations of the Shift reference flux.

If the matplotlib module is installed these tests produce plots of the binned flux compared to the Shift reference, Monaco, and MCNP5 results. They also produce a plot of the flux ratios with the Shift reference result and a histogram plot of the standard deviation difference between the Monaco flux result and the test result in each bin.

19.2 Transmission tests

Requirements: numpy and h5py modules

These tests compare the simulated flux in a detector for neutron-only and photon-only streaming problems produced by Shift. The problem consists of a 2 mfp thick slab in a void with an incident mono-energetic beam on the left of the slab. The energy of the beam varies, along with the material number density of the slab to make it 2 mfp thick (the slab has a fixed width of 5 cm). The total flux is tallied in a 2 cm block region to the right of the slab.

The slab for the neutron-only tests consists of a single nuclide with varying number densities. The slab for the photon-only tests consists of a single element with varying number densities.

Shift runs a fixed-source problem with 1e6 particle histories for each test. Currently, a Shift reference result for the flux in the detector region has been generated in serial for each test and these values are compared to the result from each test. The flux should lie within 3 standard deviations of the reference Shift results regardless of the number of processors the problem is run.

19.3 Material scaling test

CHAPTER

TWENTY

DENOVO ACCEPTANCE TEST DESCRIPTIONS

20.1 Adjoint SN transport test

20.2 Deterministic first-collision test

20.3 Symmetry test

20.4 Two-dimensional transport test

20.5 AMPX plotting test

CHAPTER TWENTYONE

STYLE GUIDE

This chapter gives some grammar, spelling, and abbreviation standards to adopt when putting together any sort of publication, presentation, or poster related to Exnihilo.

21.1 Code/Package/Library name capitalization

First, use the following capitalization when referring to code, package, and library names (in alphabetical order for easy reference).

Table 21.1: Capitalization Standards of Entities

Exnihilo package	Code	Library	Package	Program	Machine
Denovo	ADVANTG	BLAS	Atlas	CASL	Remus
Exnihilo	AMPX	LAPACK	Kokkos	VERA	Romulus
Insilico	DagMC	SuperLU	ORIGEN	OLCF	Summit
Nemesis	KENO	TriBITS	XSProc	OIC	Titan
Omnibus	Lava				
Shift	MCNP				
Transcore	MPACT				
	OpenMC				
	Profugus				
	SCALE				
	Trilinos				
	VESTA				

For a list and capitalization standard of all Trilinos packages, please see its online documentation: http://trilinos.org/packages/

21.2 Commonly used abbreviations/acronyms

Second, the following table gives abbreviations and acronyms we commonly use when referring to terms, methods, or packages.

Table 21.2: Common Abbreviations

Abbreviation	What does it mean
API	Application Programming Interface
CSV	Comma-Separated Value
CADIS	Consistent Adjoint-Driven Importance Sampling
CDF	Cumulative Distribution Function
CPU	Central Processing Unit
ENDF	Evaluated Nuclear Data Files
GMRES	Generalized Minimum Residual
GPU	General Processing Unit
HDF5	Hierarchical Data Format 5
HPC	High-Performance Computing
HZP	Hot Zero Power
KERMA	Kinetic Energy Released per Unit Mass
LWR	Light Water Reactor
MC	Monte Carlo
MG	Multigroup
MOC	Method of Characteristics
MSOD	Multiple-Set Overlapping Domain
PWR	Pressurized Water Reactor
RMS	Root Mean Squared
RTK	Reactor ToolKit
SCE	SCALE Continuous Energy
SMG	SCALE Multigroup
XML	Extended Markup Language

21.3 Commonly Used Terminology

Third, the following words or phrases are commonly used in our publications and these are the standards we have adopted when using them.

Table 21.3: Common Abbreviations

Term	Usage	Example
continuous-energy	adjective	I love continuous-energy data.
domain-decomposed, **domain-replicated**	adjective	We support domain-decomposed and domain-replicated geometries.
front end	noun	The Omnibus front end is awesome!
high-performance	adjective	Use high-performance computers.
intra-set, **intra-block**	adjective	Our code uses intra-set and intra-block communication.
massively parallel	adjective	We have a massively parallel code.
multigroup	adjective	I also love multigroup data.
multiset	adjective	Multiset decomposition is best.
object-oriented	adjective	Only use object-oriented languages.
path length	noun	Tallying path length can be hard.
pincell	noun	Reactor assemblies have pincells.
preprocess , **postprocess**	adjective, verb	We can use Omnibus to preprocess and postprocess results.
run time	noun	Parameters are set at run time.

CHAPTER

TWENTYTWO

LICENSE INFORMATION

The following licenses may apply to code distributed with Exnihilo.

22.1 Trilinos

BSD code in the Trilinos library licensed by Sandia contains the following notice:

```
Under terms of Contract DE-AC04-94AL85000, there is a non-exclusive
license for use of this work by or on behalf of the U.S. Government.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

1. Redistributions of source code must retain the above copyright
notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.

3. Neither the name of the Corporation nor the names of the
contributors may be used to endorse or promote products derived from
this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY SANDIA CORPORATION "AS IS" AND ANY
EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL SANDIA CORPORATION OR THE
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

22.2 Google Test

Google Test is a test harness used by Exnihilo.

CHAPTER

TWENTYTHREE

ACKNOWLEDGMENTS

23.1 Copyright